
Simba

Simba Documentation

Release 11.0.0

Erik Moqvist

Nov 29, 2016

1	Videos	3
2	Features	341
3	Testing	343
4	Design goals	345
5	Indices and tables	347
	Python Module Index	349

Simba is an Embedded Programming Platform. It aims to make embedded programming easy and portable.

Project homepage: <https://github.com/erimoq/simba>

Videos

Transmit CAN frames between a Nano32 and an Arduino Due. More videos are available on the [Videos](#) page.

1.1 Installation

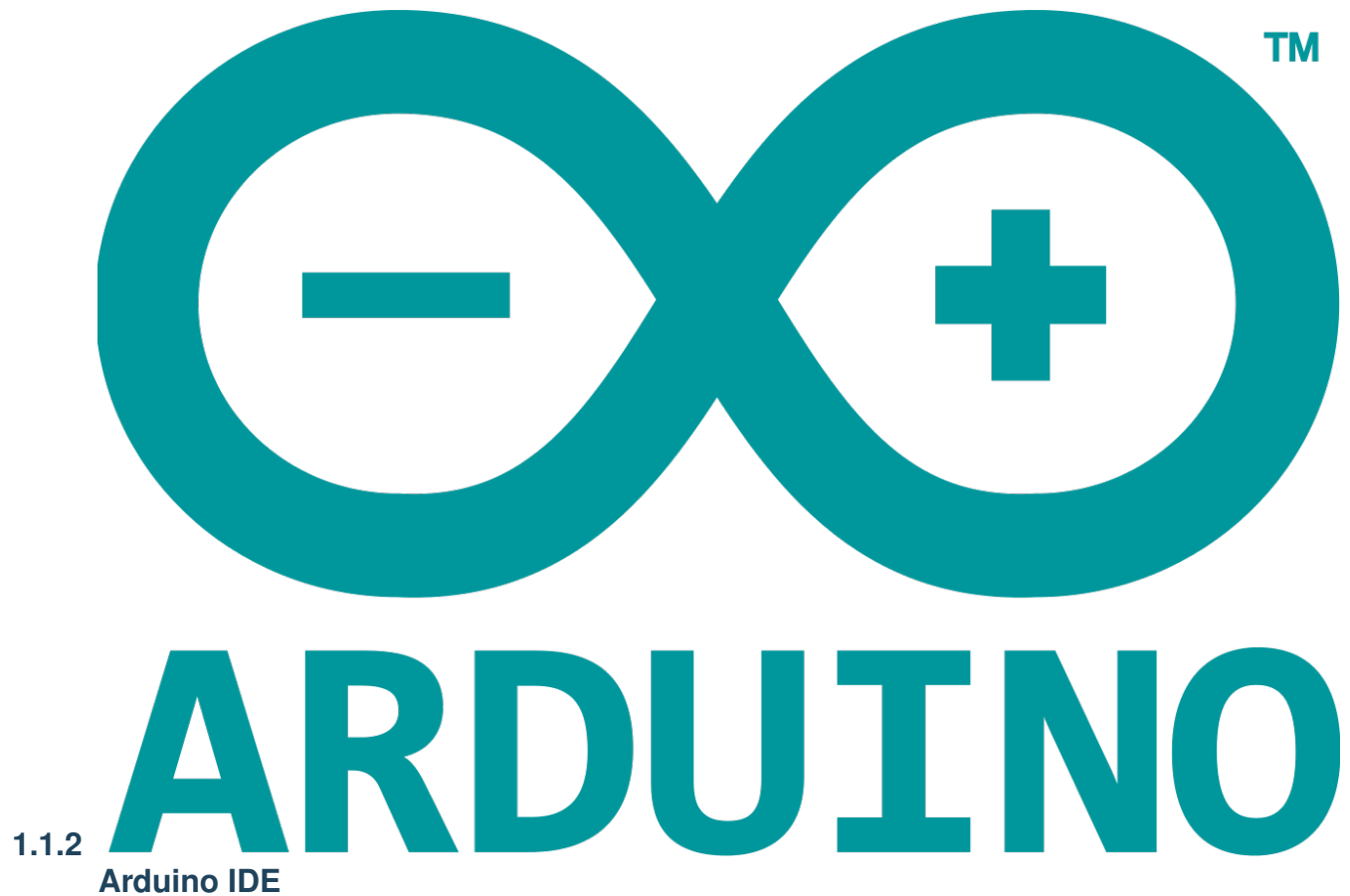
There are three build systems available; *PlatformIO*, *Arduino IDE* and *Simba build system*. The *Simba build system* has more features than the other two. It supports executing test suites, generating code coverage, profiling and more. Still, if you are familiar with *Arduino IDE* or *PlatformIO*, use that instead since it will be less troublesome.



1.1.1

PlatformIO

Install *Simba* in [PlatformIO](#).



Install *Simba* in the [Arduino IDE 1.6.10](#) as a third party board using the Boards Manager.

1. Open *File* -> *Preferences*.
2. Add these URL:s to *Additional Boards Manager URLs* (click on the icon to the right of the text field) and press *OK*.

```
https://raw.githubusercontent.com/erimoq/simba-releases/master/arduino/avr/  
→package_simba_avr_index.json  
https://raw.githubusercontent.com/erimoq/simba-releases/master/arduino/sam/  
→package_simba_sam_index.json  
https://raw.githubusercontent.com/erimoq/simba-releases/master/arduino/esp/  
→package_simba_esp_index.json  
https://raw.githubusercontent.com/erimoq/simba-releases/master/arduino/esp32/  
→package_simba_esp32_index.json
```

3. Open *Tools* -> *Board: ...* -> *Boards Manager...* and type *simba* in the search box.
4. Click on *Simba by Erik Moqvist version x.y.z* and click *Install* and press *Close*.
5. Open *Tools* -> *Board: ...* -> *Boards Manager...* and select one of the Simba boards in the list.
6. Open *File* -> *Examples* -> *Simba* -> *hello_world*.
7. Verify and upload the sketch to your device.
8. Done!

Simba

1.1.3 Simba build system

The *Simba* development environment can be installed on *Windows (Cygwin)* and *Linux (Ubuntu 14)*. Just follow the steps below and you'll be up and running in no time. =)

Windows (Cygwin)

Download [Cygwin](#) and select the following packages for installation:

NOTE: ESP8266 is not supported in *Cygwin* because there is no toolchain available.

```
- gcc-core          (Devel -> gcc-core)
- make              (Devel -> make)
- python            (Python -> python)
- python-setuptools (Python -> python-setuptools)
- git               (Devel -> git)
- doxygen           (Devel -> doxygen)
```

Start *Cygwin* and execute the one-liner below to install *Simba*.

```
$ mkdir simba && \
  cd simba && \
  easy_install-2.7 pip && \
  pip install pyserial xpect readchar sphinx breathe && \
  git clone https://github.com/erimoq/avr-toolchain-windows && \
  git clone https://github.com/erimoq/arm-toolchain-windows && \
  git clone https://github.com/erimoq/simba
```

Linux (Ubuntu 14)

Execute the one-liner below to install *Simba*.

```
$ mkdir simba && \
  cd simba && \
  sudo apt install ckermit valgrind cppcheck cloc python python-pip doxygen git lcov &
↪ & \
```

```
sudo apt install avrdude gcc-avr binutils-avr gdb-avr avr-libc && \
sudo apt install bossa-cli gcc-arm-none-eabi && \
sudo apt install make unrar autoconf automake libtool gcc g++ gperf \
    flex bison texinfo gawk ncurses-dev libexpat-dev \
    python-serial sed libtool-bin pmccabe && \
sudo pip install pyserial xpect readchar sphinx breathe sphinx_rtd_theme && \
(git clone --recursive https://github.com/pfalcon/esp-open-sdk && \
 cd esp-open-sdk && \
 make) && \
wget https://github.com/eerimoq/simba-releases/raw/master/arduino/esp32/tools/
→xtensa-esp32-elf-linux$(getconf LONG_BIT)-1.22.0-59.tar.gz && \
tar xf xtensa-esp32-elf-linux$(getconf LONG_BIT)-1.22.0-59.tar.gz && \
rm xtensa-esp32-elf-linux$(getconf LONG_BIT)-1.22.0-59.tar.gz && \
git clone --recursive https://github.com/eerimoq/simba
```

Post-install

Let's build and run the hello world application to verify that the installation was successful.

```
$ cd simba && \
source setup.sh && \
cd examples/hello_world && \
make -s run
```

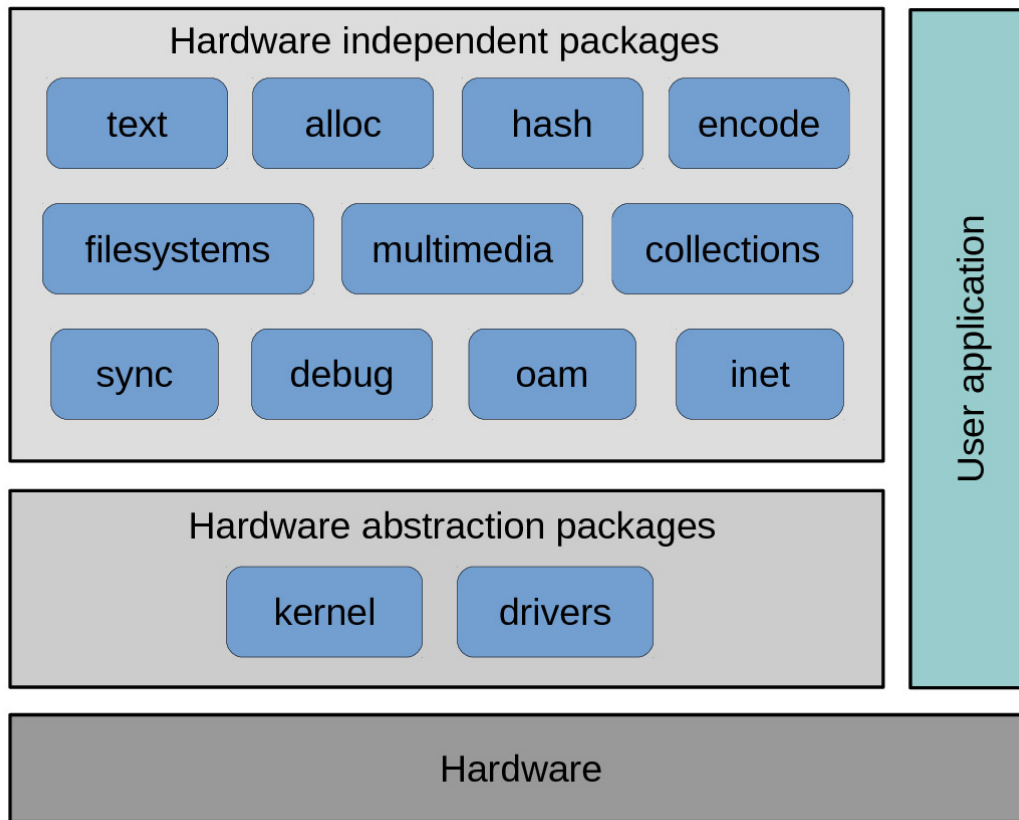
1.2 User Guide

This guide is intended for users of the Simba Embedded Programming Platform and the *Simba build system*. Parts of the guide is applicable to other build systems as well, in particular the configuration section.

The Simba installation guide can be found on the [Installation](#) page.

1.2.1 Software architecture

Below is a picture of all packages and their relation to the hardware. At the bottom is the hardware. On top of the hardware is the kernel and drivers packages, which exports a hardware independent interface that other packages and the user application can use. The user application on the right can use any package, and in rare cases directly access the hardware registers.



Contents:

Environment setup

The first step is always to setup the *Simba* environment. It's a simple matter of sourcing a setup-script in the *simba* root folder.

```
$ cd simba/simba
$ source setup.sh
```

Hello World application

Let's start with the *Simba* “Hello World” application. It exemplifies what an application is and how to build and run it. It consists of two files; `main.c` and `Makefile`.

main.c

`main.c` defines the application entry function `main()`.

```
#include "simba.h"

int main()
{
    /* Start the system. */
}
```

```
sys_start();

std_printf(FSTR("Hello world!\n"));
}
```

Makefile

Makefile contains build configuration of the application.

```
NAME = hello_world
BOARD ?= linux

RUN_END_PATTERN = "Hello world!"
RUN_END_PATTERN_SUCCESS = "Hello world!"

SIMBA_ROOT = ../..
include $(SIMBA_ROOT)/make/app.mk
```

Build and run

Compile, link and run it by typing the commands below in a shell:

```
$ cd examples/hello_world
$ make -s run
<build system output>
Hello world!
$
```

Cross-compile, link and then run on an Arduino Due:

```
$ cd examples/hello_world
$ make -s BOARD=arduino_due run
<build system output>
Hello world!
$
```

Applications, packages and modules

Simba has three software components; the application, the package and the module.

Application

An application is an executable consisting of zero or more packages.

An application file tree can either be created manually or by using the tool *simba*.

```
myapp
- main.c
- Makefile
```

Development workflow

Build and run often! More to be added, hopefully.

Package

A package is a container of modules.

A package file tree can either be created manually or by using the tool *simba*.

A package file tree **must** be organized as seen below. This is required by the build framework and *Simba* tools.

See the inline comments for details about the files and folders contents.

```

mypkg
- mypkg
|   - doc                      # package documentation
|   - __init__.py
|   - src                      # package source code
|       - mypkg
|           - module1.c
|           - module1.h
|       - mypkg.h              # package header file
|       - mypkg.mk             # package makefile
|   - tst                      # package test code
|       - module1
|           - main.c
|           - Makefile
- setup.py

```

Development workflow

The package development workflow is fairly straight forward. Suppose we want to add a new module to the file tree above. Create `src/mypkg/module2.h` and `src/mypkg/module2.c`, then include `mypkg/module2.h` in `src/mypkg.h` and add `mypkg/module2.c` to the list of source files in `src/mypkg.mk`. Create a test suite for the module. It consists of the two files `tst/module2/main.c` and `tst/module2/Makefile`.

It's often convenient to use an existing modules' files as skeleton for the new module.

After adding the module `module2` the file tree looks like this.

```

mypkg
- mypkg
|   - doc
|   - __init__.py
|   - src
|       - mypkg
|           - module1.c
|           - module1.h
|           - module2.c
|           - module2.h
|       - mypkg.h
|       - mypkg.mk
|   - tst
|       - module1
|           - main.c

```

```
|         | - Makefile
|         - module2
|         - main.c
|         - Makefile
- setup.py
```

Now, build and run the test suite to make sure the empty module implementation compiles and can be executed.

```
$ cd tst/module2
$ make -s run
```

Often the module development is started by implementing the module header file and at the same time write test cases. Test cases are not only useful to make sure the implementation works, but also to see how the module is intended to be used. The module interface becomes cleaner and easier to use if you actually start to use it yourself by writing test cases! All users of your module will benefit from this!

So, now we have an interface and a test suite. It's time to start the implementation of the module. Usually you write some code, then run the test suite, then fix the code, then run the tests again, then you realize the interface is bad, change it, change the implementation, change the test, change, change... and so it goes on until you are satisfied with the module.

Try to update the comments and documentation during the development process so you don't have to do it all in the end. It's actually quite useful for yourself to have comments. You know, you forget how to use your module too!

The documentation generation framework uses doxygen, breathe and sphinx. That means, all comments in the source code should be written for doxygen. Breathe takes the doxygen output as input and creates input for sphinx. Sphinx then generates the html documentation.

Just run make in the doc folder to generate the html documentation.

```
$ cd doc
$ make
$ firefox _build/html/index.html    # open the docs in firefox
```

Namespaces

All exported symbols in a package must have the prefix `<package>_<module>_`. This is needed to avoid namespace clashes between modules with the same name in different packages.

There cannot be two packages with the same name, for the namespace reason. All packages must have unique names! There is one exception though, the three *Simba* packages; kernel, drivers and slib. Those packages do *not* have the package name as prefix on exported symbols.

```
int mypackage_module1_foo(void);
int mypackage_module2_bar(void);
```

Module

A module is normally a header and a source file.

Configuration

Standard Library

The *Library Reference* is configured at compile time using defines named `CONFIG_*`. The default configuration includes most functionality, as most application wants that. If an application has special requirements, for example memory constraints, it has to be configured to remove unnecessary functionality.

Search order

Highest priority first.

1. Command line as `CDEFS_EXTRA="<configuration variable>=<value>"`.
2. A file named `config.h` in the application root folder.
3. The default configuration file, `src/config_default.h`.

Variables

All configuration variables are listed below. Their default values are defined in `src/config_default.h`.

Defines

CONFIG_SYS_CONFIG_STRING

CONFIG_SYS_SIMBA_MAIN_STACK_MAX

Main thread stack size for ports with a fixed size main thread stack.

CONFIG_ASSERT

Assertions are used to check various conditions during the application execution. A typical usage is to validate function input arguments.

CONFIG_DEBUG

Include more debug information.

CONFIG_FS_CMD_DS18B20_LIST

Debug file system command to list all DS18B20 sensors on the bus.

CONFIG_FS_CMD_ESP_WIFI_STATUS

Debug file system command to print the Espressif WiFi status.

CONFIG_FS_CMD_FS_APPEND

Debug file system command to append to a file.

CONFIG_FS_CMD_FS_COUNTERS_LIST

Debug file system command to list all counters.

CONFIG_FS_CMD_FS_COUNTERS_RESET

Debug file system command to set all counters to zero.

CONFIG_FS_CMD_FS_FILESYSTEMS_LIST

Debug file system command to list all registered file systems.

CONFIG_FS_CMD_FS_LIST

Debug file system command to list all registered file systems.

CONFIG_FS_CMD_FS_FORMAT

Debug file system command to format a file system.

CONFIG_FS_CMD_FS_PARAMETERS_LIST

Debug file system command to list all parameters.

CONFIG_FS_CMD_FS_READ

Debug file system command to read from a file.

CONFIG_FS_CMD_FS_REMOVE

Debug file system command to remove a file.

CONFIG_FS_CMD_FS_WRITE

Debug file system command to write to a file.

CONFIG_FS_CMD_I2C_READ

Debug file system command to read from a i2c bus.

CONFIG_FS_CMD_I2C_WRITE

Debug file system command to write to a i2c bus.

CONFIG_FS_CMD_LOG_LIST

Debug file system command to list all log objects.

CONFIG_FS_CMD_LOG_PRINT

Debug file system command to create a log entry and print it. Mainly used for debugging.

CONFIG_FS_CMD_LOG_SET_LOG_MASK

Debug file system command to set the log mask of a log object.

CONFIG_FS_CMD_NETWORK_INTERFACE_LIST

Debug file system command to list all network interfaces.

CONFIG_FS_CMD_PIN_READ

Debug file system command to read the current value of a pin.

CONFIG_FS_CMD_PIN_SET_MODE

Debug file system command to set the mode of a pin.

CONFIG_FS_CMD_PIN_WRITE

Debug file system command to write a value to a pin.

CONFIG_FS_CMD_PING_PING

Debug file system command to ping a host.

CONFIG_FS_CMD_SERVICE_LIST

Debug file system command to list all services.

CONFIG_FS_CMD_SERVICE_START

Debug file system command to start a service.

CONFIG_FS_CMD_SERVICE_STOP

Debug file system command to stop a services.

CONFIG_FS_CMD_SETTINGS_LIST

Debug file system command to list all settings.

CONFIG_FS_CMD_SETTINGS_READ

Debug file system command to read the value of a setting.

CONFIG_FS_CMD_SETTINGS_RESET

Debug file system command to reset the settings to their original values.

CONFIG_FS_CMD_SETTINGS_WRITE

Debug file system command to write a value to a setting.

CONFIG_FS_CMD_SYS_CONFIG

Debug file system command to print the system configuration.

CONFIG_FS_CMD_SYS_INFO

Debug file system command to print the system information.

CONFIG_FS_CMD_SYS_UPTIME

Debug file system command to print the system uptime.

CONFIG_FS_CMD_SYS_REBOOT

Debug file system command to reboot the system uptime.

CONFIG_FS_CMD_THRD_LIST

Debug file system command to list threads' information.

CONFIG_FS_CMD_THRD_SET_LOG_MASK

Debug file system command to set the log mask of a thread.

CONFIG_FS_CMD_USB_DEVICE_LIST

Debug file system command to list all USB devices.

CONFIG_FS_CMD_USB_HOST_LIST

Debug file system command to list all USB devices connected to the USB host.

CONFIG_FS_PATH_MAX

The maximum length of an absolute path in the file system.

CONFIG_MONITOR_THREAD

Start the monitor thread to gather statistics of the scheduler.

CONFIG_PREEMPTIVE_SCHEDULER

Use a preemptive scheduler.

CONFIG_PROFILE_STACK

Profile the stack usage in runtime. It's a cheap operation and is recommended to have enabled.

CONFIG_SETTINGS_AREA_SIZE

Size of the settings area. This size *MUST* have the same size as the settings generated by the settings.py script.

CONFIG_SHELL_COMMAND_MAX

Maximum number of characters in a shell command.

CONFIG_SHELL_HISTORY_SIZE

Size of the shell history buffer.

CONFIG_SHELL_MINIMAL

Minimal shell functionality to minimize the code size of the shell module.

CONFIG_SHELL_PROMPT

The shell prompt string.

CONFIG_SOCKET_RAW

Raw socket support.

CONFIG_SPIFFS

SPIFFS is a flash file system applicable for boards that has a reasonably big modifiable flash.

CONFIG_START_CONSOLE

Start the console device (UART/USB CDC) on system startup.

CONFIG_START_CONSOLE_DEVICE_INDEX

Console device index.

CONFIG_START_CONSOLE_UART_BAUDRATE

Console UART baudrate.

CONFIG_START_CONSOLE_USB_CDC_CONTROL_INTERFACE

Console USB CDC control interface number.

CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_IN

Console USB CDC input endpoint.

CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_OUT

Console USB CDC output endpoint.

CONFIG_START_CONSOLE_USB_CDC_WAIT_FOR_CONNENTION

Wait for the host to connect after starting the console.

CONFIG_START_FILESYSTEM

Configure a default file system.

CONFIG_START_FILESYSTEM_ADDRESS

Configure a default file system start address.

CONFIG_START_FILESYSTEM_SIZE

Configure a default file system size.

CONFIG_START_NETWORK

Setup the ip stack and connect to all configured networks.

CONFIG_START_NETWORK_INTERFACE_WIFI_CONNECT_TIMEOUT

WiFi connect timeout is seconds.

CONFIG_START_NETWORK_INTERFACE_WIFI_SSID

SSID of the WiFi to connect to.

CONFIG_START_NETWORK_INTERFACE_WIFI_PASSWORD

Password of the WiFi to connect to.

CONFIG_START_SHELL

Start a shell thread communication over the console channels.

CONFIG_START_SHELL_PRIO

Shell thread priority.

CONFIG_START_SHELL_STACK_SIZE

Shell thread stack size in words.

CONFIG_STD_OUTPUT_BUFFER_MAX

Maximum number of bytes in the print output buffer.

CONFIG_SYSTEM_TICK_FREQUENCY

System tick frequency in Hertz.

CONFIG_THRD_CPU_USAGE

Calculate thread CPU usage.

CONFIG_THRD_ENV

Each thread has a list of environment variables associated with it. A typical example of an environment variable is "CWD" - Current Working Directory.

CONFIG_THRD_IDLE_STACK_SIZE

Stack size of the idle thread.

CONFIG_THRD_SCHEDULED

Count the number of times each thread has been scheduled.

CONFIG_THRD_TERMINATE

Threads are allowed to terminate.

CONFIG_USB_DEVICE_VID

USB device vendor id.

CONFIG_USB_DEVICE_PID

USB device product id.

lwIP

Use `config.h` to fully configure lwIP and all of its modules. You do not need to define every option that lwIP provides; if you do not define an option, a default value will be used. Therefore, your `config.h` provides a way to override much of the behavior of lwIP.

By default *Simba* overrides a few of the variables in `src/inet/lwipopts.h`.

Module support (Code size)**Enabling and disabling modules**

You can tune your code size by only compiling the features you really need. The following is a list of what gets compiled in “out of the box” with lwIP.

Default inclusions:

- ARP (LWIP_ARP)
- IP and fragmentation (IP_FRAG) and reassembly (IP_REASSEMBLY)
- Raw IP PCB support (LWIP_RAW)
- UDP (LWIP_UDP) and UDP-Lite (LWIP_UDPLITE)
- TCP (LWIP_TCP) – this is a big one!
- Statistics (LWIP_STATS)

Default exclusions:

- DHCP (LWIP_DHCP)
- AUTOIP (LWIP_AUTOIP)
- SNMP (LWIP_SNMP)
- IGMP (LWIP_IGMP)
- PPP (PPP_SUPPORT)

If you would like to change this, then you just need to set the options listed below. For example, if you would like to disable UDP and enable DHCP, the following `config.h` file would do it:

```
/* Disable UDP */
#define LWIP_UDP 0

/* Enable DHCP */
#define LWIP_DHCP 1
```

Memory management (RAM usage)

Memory pools

In an embedded environment, memory pools make for fast and efficient memory allocation. lwIP provides a flexible way to manage memory pool sizes and organization.

lwIP reserves a fixed-size static chunk of memory in the data segment, which is subdivided into the various pools that lwIP uses for the various data structures. For example, there is a pool just for struct tcp_pcb's, and another pool just for struct udp_pcb's. Each pool can be configured to hold a fixed number of data structures; this number can be changed in the `config.h` file by changing the various `MEMP_NUM_*` values. For example, `MEMP_NUM_TCP_PCB` and `MEMP_NUM_UDP_PCB` control the maximum number of tcp_pcb and udp_pcb structures that can be active in the system at any given time.

It is also possible to create custom memory pools in addition to the standard ones provided by lwIP.

Dynamic allocation: mem_malloc

lwIP uses a custom function `mem_malloc` for all dynamic allocation; therefore, it is easy to change how lwIP uses its RAM. There are three possibilities provided out-of-the-box:

1. (default) lwIP's custom heap-based `mem_malloc`. By default, lwIP uses a statically-allocated chunk of memory like a heap for all memory operations. Use `MEM_SIZE` to change the size of the lwIP heap.
2. C standard library `malloc` and `free`. If you wish to have lwIP use the standard library functions provided by your compiler/architecture, then define the option `MEM_LIBC_MALLOC`.
3. Memory pools. lwIP can also emulate dynamic allocation using custom memory pools (see that chapter for more information). This involves the options `MEM_USE_POOLS` and `MEMP_USE_CUSTOM_POOLS` and a new custom file `lwippools.h`.

Understanding/changing memory usage

lwIP uses memory for:

- code (depending on your system, may use ROM instead of RAM)
- statically allocated variables (some initialized, some not initialized)
- task stack
- dynamically allocated memory
 - heap
 - memp pools

Unless you use a C library heap implementation (by defining `MEM_LIBC_MALLOC` to 1), dynamically allocated memory must be statically allocated somewhere. This means you reserve a specific amount of memory for the heap or the memp pools from which the code dynamically allocates memory at runtime.

The size of this heap and memp pools can be adjusted to save RAM:

There are 3 types of pbufs:

- REF/ROM, RAM and POOL. `PBUF_POOL_SIZE * PBUF_POOL_BUFSIZE` only refers to type POOL.
- RAM pbufs are allocated in the memory defined by `MEM_SIZE` (this memory is not used much aside from RAM pbufs) - this is the *heap* and it is allocated as `mem_memory`.

- REF/ROM pbufs as well as pcbs and some other stuff is allocated from dedicated pools per structure type. The amount of structures is defined by the various `MEMP_NUM_` defines. Together, this memory is allocated as `memp_memory` and it *includes* the pbuf POOL.

However, if you define `MEMP_MEM_MALLOC` to 1 in your `config.h`, *every* piece of dynamically allocated memory will come from the heap (the size of which is defined by `MEM_SIZE`). If you then even define `MEM_LIBC_MALLOC` to 1, too, lwIP doesn't need extra memory for dynamically allocated memory but only uses the C library heap instead. However, you then have to make sure that this heap is big enough to run your application.

To tweak the various `MEMP_NUM_` defines, define `LWIP_STATS=1` and `LWIP_STATS_DISPLAY=1` and call `stats_display()` to see how many entries of each pool are used (or have a look at the global variable `lwip_stats` instead).

Fine-tuning even more

To see the options that you can set, open [3pp/lwip-1.4.1/src/include/lwip/opt.h](#). This file is fully commented and explains how many of the options are used.

Build system

The *Simba* build system is based on *GNU Make*.

Targets

Name	Description
all	Compile and link the application.
clean	Remove all generated files and folders.
new	clean + all
upload	all + Upload the application to the device.
console	Open a serial console on /dev/arduino with baudrate BAUDRATE.
run	all + upload + Wait for application output.
run-debugger	Run the application in the debugger, break at main.
report	Print the test report from a previous run.
test	run + report
release	Compile with <code>NASSERT=yes</code> and <code>NDEBUG=yes</code> .
size	Print application size information.
help	Show the help.

Variables

There are plenty of make variables used to control the build process. Below is a list of the most frequently used variables. The advanced user may read the make files in [make](#).

Name	Description
SIMBA_ROOT	Path to the <i>Simba</i> root folder.
BOARD	The BOARD variable selects which board to build for. It can be assigned to one of the boards listed here . For example, the command to build for <i>Arduino Due</i> is <code>make BOARD=arduino_due</code> .
BAUDRATE	Serial port baudrate used by console and run targets.
VERSION	The application version string. Usually on the form <major>.<minor>.<revision>.
SETTINGS_INI	Path to the settings file.
INC	Include paths.
SRC	Source files (.c, .asm, .rs).
CFLAGS_EXTRA	Extra flags passed to the compiler.
LD_FLAGS_EXTRA	Extra flags passed to the linker.
NASSERT	Build the application without assertions.

simba

The program *simba* is used to manage *Simba* packages and applications.

The main purpose of *simba* is to distribute software in the *Simba* community, just like *pip* for Python.

How to create an application skeleton

The code block below shows how to create a new application using *simba*. After the application has been created, it is built and executed.

```
$ mkdir myapp
$ cd myapp
$ simba application init
Application name [foo]: <Enter>
Author [erik]: <Enter>
Version [0.3.0]: <Enter>
$ tree .
.
- main.c
- Makefile
$ make -s run
```

How to create a package

The code block below shows how to create a new package using *simba*. After the package has been created, the generated test suite is built and executed.

```
$ mkdir mypkg
$ cd mypkg
$ simba package init
Package name [foo]: <Enter>
Author [erik]: <Enter>
Version [0.3.0]: <Enter>
$ tree
.
- mypkg
```

```

|   - doc
|   |   - about.rst
|   |   - api-reference.rst
|   |   - conf.py
|   |   - doxygen.cfg
|   |   - index.rst
|   |   - Makefile
|   |   - mypkg
|   |   |   - hello.rst
|   |   - requirements.txt
|   |   - sphinx.mk
|   - __init__.py
|   - src
|   |   - mypkg
|   |   |   - hello.c
|   |   |   - hello.h
|   |   - mypkg.h
|   |   - mypkg.mk
|   - tst
|       - hello
|           - main.c
|           - Makefile
- setup.py
$ cd mypkg/tst/hello
$ make -s test

```

In the output from `tree` below, two files may catch your eyes; `setup.py` and `__init__.py`. Those are Python files and are often seen in Python packages. They are present in a *Simba* package because *Simba* uses the Python tool *pip* to release and install packages. The idea is that everyone that implements a useful package should release it and make it available for other users to install, just as Python!

How to release a package

This is how to release a package. Two files are created, one with the suffix `tar.gz` and one with the suffix `.whl`. The `.whl`-file is input to the installation command, described in the next section.

```

$ cd ../../..
$ simba package release
$ tree dist
dist
- mypkg-0.1-py2.py3-none-any.whl
- mypkg-0.1.tar.gz

```

How to install a package

This is how to install a package in `${SIMBA_ROOT}/dist-packages`.

```

$ simba package install dist/mypkg-0.1-py2.py3-none-any.whl

```

1.3 Developer Guide

This guide is intended for developers of the Simba Embedded Programming Platform. Users are advised to read the *User Guide* instead.

Contents:

1.3.1 Boards and mcus

A board is the top level configuration entity in the build framework. It contains information about the MCU and the pin mapping.

In turn, the MCU contains information about available devices and clock frequencies in the microcontroller.

See [src/boards/](#) and [src/mcus](#) for available configurations.

Only one MCU per board is supported. If there are two MCU:s on one physical board, two board configurations have to be created, one for each MCU.

The porting guide [Porting](#) shows how to port *Simba* to a new board.

1.3.2 Threads and channels

A thread is the basic execution entity. A scheduler controls the execution of threads.

A simple thread that waits to be resumed by another thread.

```
#include "simba.h"

void *my_thread_main(void *arg_p)
{
    UNUSED(arg_p);

    while (1) {
        thrd_suspend(NULL);
        std_printf(FSTR("Thread resumed.\r\n"));
    }

    return (NULL);
}
```

Threads usually communicates over channels. There are two kinds of channels; queue and event. Both implementing the same abstract channel interface (see [src/kernel/chan.h](#)). This abstraction makes channel very powerful as a synchronization primitive. They can be seen as limited functionality file descriptors in linux.

The most common channel is the queue. It can be either synchronous or semi-asynchronous. In the synchronous version the writing thread will block until all written data has been read by the reader. In the semi-asynchronous version the writer writes to a buffer within the queue, and only blocks all data does not fit in the buffer. The buffer size is selected by the application.

1.3.3 File tree

simba	- this directory
- 3pp	- third party products
- bin	- executables and scripts

- doc	- documentation source
- environment	- environment setup
- examples	- example applications
- LICENSE	- license
- make	- build and run files
- README.rst	- readme
- setup.sh	- setup script
- src	- source code directory
- alloc	- alloc package
- boards	- board configurations
- collections	- collections package
- debug	- debug package
- drivers	- drivers package
- encode	- encode package
- filesystems	- filesystems package
- hash	- hash package
- inet	- inet package
- kernel	- kernel package
- mcus	- mcu configurations
- multimedia	- multimedia package
- oam	- oam package
- sync	- sync package
- text	- text package
- simba.h	- includes all package headers
- simba.mk	- build system configuration
- tst	- test suites
- alloc	- alloc package test suite
- collections	- collections package test suite
- debug	- debug package test suite
- drivers	- drivers package test suite
- encode	- encode package test suite
- filesystems	- filesystems package test suite
- hash	- hash package test suite
- inet	- inet package test suite
- kernel	- kernel package test suite
- multimedia	- multimedia package test suite
- oam	- oam package test suite
- sync	- sync package test suite
- text	- text package test suite
- VERSION.txt	- `Simba` version

1.3.4 Testing

To ensure high code quality each module is tested extensively by many test suites. The test suites are executed both on native Linux and on many of the supported boards. See *Test suites* for a list of all test suites that are executed before each release.

The native Linux test suites are executed automatically on each commit.

Test result: <https://travis-ci.org/eerimoq/simba>

Code coverage: <https://codecov.io/gh/eerimoq/simba>

Hardware setup

Below is a picture of all supported boards connected to a USB hub. The USB hub is connected to a linux PC (not in the picture) that executes test suites on all boards.

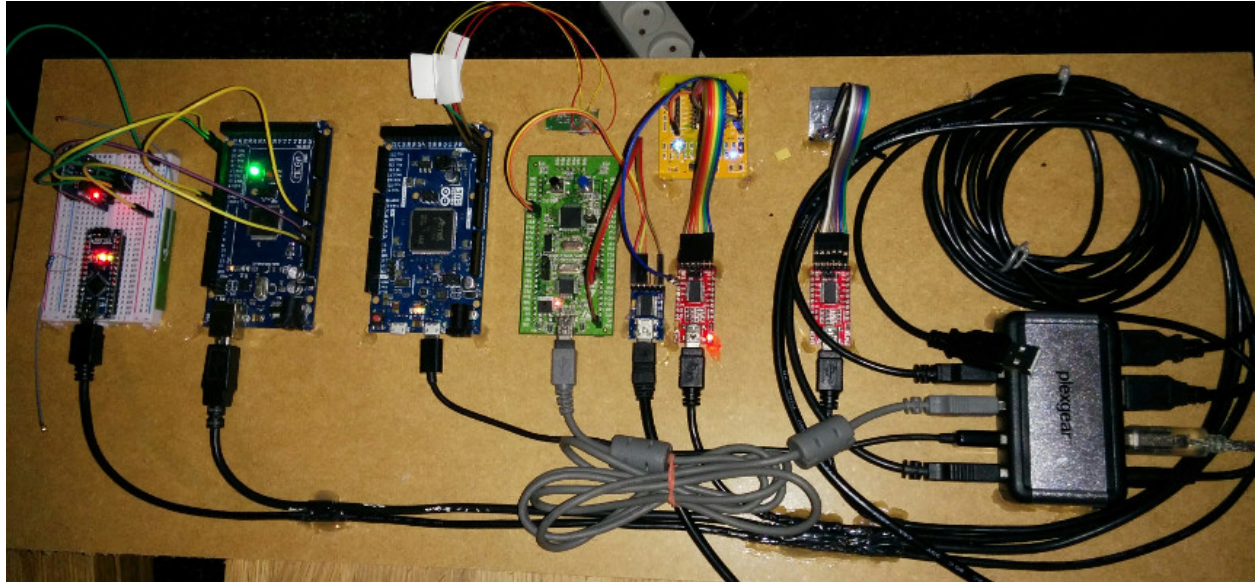


Fig. 1.1: The boards are (from left to right): *Arduino Nano*, *Arduino Mega*, *Arduino Due*, *STM32VLDISCOVERY*, *ESP-12E Development Board* and *ESP-01*

A short description of the setup:

- The DS3231 device (on the breadboard to the left) is connected over i2c to the *Arduino Mega*.
- CAN0 is connected to CAN1 on the *Arduino Due*. The CAN driver is tested by sending frames between the two CAN devices.
- The UART of the *STM32VLDISCOVERY* board is connected to a serial to USB adaptor. DTR on the adaptor is used to reset the board.
- The *ESP-12E Development Board* also has a serial to USB adaptor connected. RTS is used to set the board in flashing mode (GPIO0) and DTR is used to reset the board (REST).

Test suites

Below is a list of all test suites that are executed before every release. They are listed per board.

Arduino Due

- `kernel/sys`
- `kernel/thrd`
- `kernel/time`
- `kernel/timer`
- `sync/bus`

- `sync/event`
- `sync/queue`
- `sync/rwlock`
- `sync/sem`
- `collections/binary_tree`
- `collections/bits`
- `collections/fifo`
- `collections/hash_map`
- `alloc/circular_heap`
- `alloc/heap`
- `text/configfile`
- `text/std`
- `text/re`
- `debug/log`
- `oam/settings`
- `oam/shell`
- `filesystems/fs`
- `filesystems/spiffs`
- `encode/base64`
- `encode/json`
- `hash/crc`
- `hash/sha1`
- `inet/http_server`
- `inet/http_websocket_client`
- `inet/http_websocket_server`
- `inet/inet`
- `inet/mqtt_client`
- `inet/ping`
- `drivers/chipid`
- `drivers/can`
- `drivers/flash`
- `drivers/pin`

Arduino Mega

- kernel/sys
- kernel/thrd
- kernel/time
- kernel/timer
- sync/bus
- sync/event
- sync/queue
- sync/rwlock
- sync/sem
- collections/binary_tree
- collections/bits
- collections/fifo
- collections/hash_map
- alloc/circular_heap
- alloc/heap
- text/configfile
- text/std
- text/re
- debug/log
- oam/settings
- oam/shell
- filesystems/fat16
- filesystems/fs
- encode/base64
- hash/crc
- hash/sha1
- inet/http_websocket_client
- inet/http_websocket_server
- inet/inet
- inet/mqtt_client
- inet/ping
- drivers/adc
- drivers/analog_input_pin
- drivers/ds3231
- drivers/sd

- `drivers/pin`

Arduino Nano

- `drivers/ds18b20`
- `drivers/analog_output_pin`
- `drivers/exti`
- `drivers/owi`

Arduino Pro Micro

- `kernel/sys`
- `kernel/thrd`
- `kernel/timer`

Arduino Uno

Cygwin

ESP-01

- `kernel/sys`
- `kernel/thrd`
- `kernel/timer`

ESP-12E Development Board

- `kernel/sys`
- `kernel/thrd`
- `kernel/timer`
- `sync/bus`
- `sync/event`
- `sync/queue`
- `sync/rwlock`
- `sync/sem`
- `collections/binary_tree`
- `collections/bits`
- `collections/fifo`
- `collections/hash_map`
- `alloc/circular_heap`
- `text/std`

- text/re
- debug/log
- oam/shell
- encode/base64
- encode/json
- hash/crc
- hash/sha1
- inet/http_websocket_client
- inet/http_websocket_server
- inet/inet
- inet/mqtt_client
- inet/network_interface/wifi_esp
- inet/ping
- drivers/pin
- filesystems/fs
- filesystems/spiffs

Linux

- kernel/sys
- kernel/thrd
- kernel/time
- kernel/timer
- sync/bus
- sync/chan
- sync/event
- sync/queue
- sync/rwlock
- sync/sem
- collections/binary_tree
- collections/bits
- collections/fifo
- collections/hash_map
- alloc/circular_heap
- alloc/heap
- text/configfile
- text/std

- text/re
- debug/log
- oam/service
- oam/settings
- oam/shell
- filesystems/fat16
- filesystems/fs
- filesystems/spiffs
- encode/base64
- encode/json
- hash/crc
- hash/sha1
- inet/http_server
- inet/http_websocket_client
- inet/http_websocket_server
- inet/inet
- inet/mqtt_client
- inet/ping
- multimedia/midi

Nano32

- kernel/sys
- kernel/thrd
- kernel/timer
- sync/bus
- sync/event
- sync/queue
- sync/rwlock
- sync/sem
- collections/binary_tree
- collections/bits
- collections/fifo
- collections/hash_map
- alloc/circular_heap
- text/std
- text/re

- debug/log
- oam/shell
- encode/base64
- encode/json
- hash/crc
- hash/sha1
- inet/http_websocket_client
- inet/http_websocket_server
- inet/inet
- inet/network_interface/wifi_esp
- inet/ping
- filesystems/fs
- filesystems/spiffs

Particle IO Photon

- kernel/sys
- kernel/thrd
- kernel/time
- kernel/timer
- sync/bus
- sync/event
- sync/queue
- sync/rwlock
- sync/sem
- collections/binary_tree
- collections/bits
- collections/fifo
- collections/hash_map
- alloc/circular_heap
- text/std
- text/re
- debug/log
- oam/shell
- encode/base64
- encode/json
- hash/crc

- hash/sha1
- inet/http_websocket_client
- inet/http_websocket_server
- inet/inet
- inet/mqtt_client
- inet/ping

STM32F3DISCOVERY

STM32VLDISCOVERY

- kernel/sys
- kernel/thrd
- kernel/timer
- sync/bus
- sync/event
- sync/queue
- sync/rwlock
- sync/sem
- collections/binary_tree
- collections/bits
- collections/fifo
- collections/hash_map
- alloc/circular_heap
- text/std
- text/re
- debug/log
- oam/shell
- encode/base64
- encode/json
- hash/crc
- hash/sha1
- inet/http_websocket_client
- inet/http_websocket_server
- inet/inet
- inet/mqtt_client
- inet/ping
- drivers/pin

1.3.5 Releasing

Follow these steps to create a new release:

1. Write the new version in `VERSION.txt`. The version should have the format `<major>.<minor>.<revision>`.

Increment `<major>` for non-backwards compatible changes.

Increment `<minor>` for new features.

Increment `<revision>` for bug fixes.

2. Write the new version in `package.json`. This file is used by *PlatformIO 3* to find the current *Simba* release.
3. Run the test suites and generate the documentation.

```
make -s test-all-boards
make -s release-test
```

4. Generate files for Arduino.

```
make -s arduino
```

5. Add the new releases to `make/arduino/<family>/package_simba_<family>_index.json`. The sha256 sums of the zip-archives are calculated by `make -s arduino` and written to `simba-arduino/*.sha256`.

6. Copy the Simba Arduino releases to the release repository, add, commit and push in the release repository.

```
cp simba-arduino/simba-arduino-avr-*.zip ../simba-releases/arduino/avr
cp simba-arduino/simba-arduino-sam-*.zip ../simba-releases/arduino/sam
cp simba-arduino/simba-arduino-esp-*.zip ../simba-releases/arduino/esp
cp simba-arduino/simba-arduino-esp32-*.zip ../simba-releases/arduino/esp32
```

7. Start a http server used to download package manifests in the Arduino IDE.

```
(cd make/arduino && python -m SimpleHTTPServer)
```

8. Start the Arduino IDE and add these URL:s in Preferences.

```
http://localhost:8000/avr/package_simba_avr_index.json
http://localhost:8000/esp/package_simba_esp_index.json
http://localhost:8000/esp32/package_simba_esp32_index.json
http://localhost:8000/sam/package_simba_sam_index.json
```

9. Install all four packages and run the blink example for each one of them.
10. Commit the changes, and tag the commit with the new version.
11. Push the new commit and tag.
12. Copy the Simba Arduino package manifests the release repository, add, commit and push in the release repository.

```
cp make/arduino/avr/package_simba_avr_index.json ../simba-releases/arduino/
↪avr
cp make/arduino/sam/package_simba_sam_index.json ../simba-releases/arduino/
↪sam
cp make/arduino/esp/package_simba_esp_index.json ../simba-releases/arduino/
↪esp
cp make/arduino/esp32/package_simba_esp32_index.json ../simba-releases/
↪arduino/esp32
```

-
13. Download the release zip-file from Github and calculate its SHA1 checksum. Add the zip-file to the release repository and add the new releases to `make/platformio/manifest.json`.

```
wget https://github.com/eerimoq/simba/archive/<version>.zip
shasum <version>.zip
cp <version>.zip ../simba-releases/platformio
```

14. Commit and push.
15. Done.

1.3.6 Porting

Often the board you want to use in your project is not yet supported by *Simba*. If you are lucky, *Simba* is already ported to the MCU on your board. Just create a folder with you board name in `src/boards/` and populate it with the `board.h`, `board.c` and `board.mk`. If *Simba* is not ported to your MCU, the kernel and drivers has to be ported.

Kernel

Porting the kernel is a matter of configuring the system tick timer and implement a few locking primitives. If you are familiar with your CPU, the port can be implemented quickly.

A kernel port is roughly 300 lines of code.

Kernel ports are implemented in `src/kernel/ports`.

Drivers

The required work to port the drivers depends of which drivers you are intrested in. The more drivers you have to port, the longer time it takes, obviously.

A drivers port is roughly 100 lines of code per driver.

Drivers ports are implemented in `src/drivers/ports`.

1.4 Boards

The boards supported by *Simba*.

- *pin* — Digital pins
- *sd* — Secure Digital memory
- *spi* — Serial Peripheral Interface
- *uart* — Universal Asynchronous Receiver/Transmitter
- *usb* — Universal Serial Bus
- *usb_host* — Universal Serial Bus - Host

Library Reference

Read more about board specific functionality in the *Arduino Due* module documentation in the Library Reference.

Memory usage

Below is the memory usage of two applications:

- The *minimal-configuration* application is configured to only include the bare minimum of functionality for the low level kernel to run. That is, the thread scheduler and system tick.
- The *default-configuration* application is built with the default configuration, including a lot more functionality. See the list of *Default system features* above for a summary.

Application	Flash	RAM
minimal-configuration	21564	5440
default-configuration	94584	11018

Default configuration

Default Standard Library configuration.

Name	Value
CONFIG_ASSERT	1
CONFIG_DEBUG	1
CONFIG_FS_CMD_DS18B20_LIST	1
CONFIG_FS_CMD_ESP_WIFI_STATUS	0
CONFIG_FS_CMD_FS_APPEND	1
CONFIG_FS_CMD_FS_COUNTERS_LIST	1
CONFIG_FS_CMD_FS_COUNTERS_RESET	1
CONFIG_FS_CMD_FS_FILESYSTEMS_LIST	1
CONFIG_FS_CMD_FS_FORMAT	1
CONFIG_FS_CMD_FS_LIST	1
CONFIG_FS_CMD_FS_PARAMETERS_LIST	1
CONFIG_FS_CMD_FS_READ	1
CONFIG_FS_CMD_FS_REMOVE	1
CONFIG_FS_CMD_FS_WRITE	1
CONFIG_FS_CMD_I2C_READ	1
CONFIG_FS_CMD_I2C_WRITE	1
CONFIG_FS_CMD_LOG_LIST	1
CONFIG_FS_CMD_LOG_PRINT	1
CONFIG_FS_CMD_LOG_SET_LOG_MASK	1

Continued on next page

Table 1.1 – continued from previous page

Name	Value
CONFIG_FS_CMD_NETWORK_INTERFACE_LIST	1
CONFIG_FS_CMD_PING_PING	1
CONFIG_FS_CMD_PIN_READ	1
CONFIG_FS_CMD_PIN_SET_MODE	1
CONFIG_FS_CMD_PIN_WRITE	1
CONFIG_FS_CMD_SERVICE_LIST	1
CONFIG_FS_CMD_SERVICE_START	1
CONFIG_FS_CMD_SERVICE_STOP	1
CONFIG_FS_CMD_SETTINGS_LIST	1
CONFIG_FS_CMD_SETTINGS_READ	1
CONFIG_FS_CMD_SETTINGS_RESET	1
CONFIG_FS_CMD_SETTINGS_WRITE	1
CONFIG_FS_CMD_SYS_CONFIG	1
CONFIG_FS_CMD_SYS_INFO	1
CONFIG_FS_CMD_SYS_REBOOT	1
CONFIG_FS_CMD_SYS_UPTIME	1
CONFIG_FS_CMD_THRD_LIST	1
CONFIG_FS_CMD_THRD_SET_LOG_MASK	1
CONFIG_FS_CMD_USB_DEVICE_LIST	1
CONFIG_FS_CMD_USB_HOST_LIST	1
CONFIG_FS_PATH_MAX	64
CONFIG_MONITOR_THREAD	1
CONFIG_PREEMPTIVE_SCHEDULER	0
CONFIG_PROFILE_STACK	1
CONFIG_SETTINGS_AREA_SIZE	256
CONFIG_SHELL_COMMAND_MAX	64
CONFIG_SHELL_HISTORY_SIZE	768
CONFIG_SHELL_MINIMAL	0
CONFIG_SHELL_PROMPT	“\$ “
CONFIG_SOCKET_RAW	1
CONFIG_SPIFFS	1
CONFIG_START_CONSOLE	CONFIG_START_CONSOLE_UART
CONFIG_START_CONSOLE_DEVICE_INDEX	0
CONFIG_START_CONSOLE_UART_BAUDRATE	38400
CONFIG_START_CONSOLE_USB_CDC_CONTROL_INTERFACE	0
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_IN	2
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_OUT	3
CONFIG_START_CONSOLE_USB_CDC_WAIT_FOR_CONNETION	1
CONFIG_START_FILESYSTEM	1
CONFIG_START_FILESYSTEM_ADDRESS	0x000e0000
CONFIG_START_FILESYSTEM_SIZE	32768
CONFIG_START_NETWORK	0
CONFIG_START_NETWORK_INTERFACE_WIFI_CONNECT_TIMEOUT	30
CONFIG_START_NETWORK_INTERFACE_WIFI_PASSWORD	MyWiFiPassword
CONFIG_START_NETWORK_INTERFACE_WIFI_SSID	MyWiFiSSID
CONFIG_START_SHELL	1
CONFIG_START_SHELL_PRIO	30
CONFIG_START_SHELL_STACK_SIZE	1536
CONFIG_STD_OUTPUT_BUFFER_MAX	16

Continued on next page

Table 1.1 – continued from previous page

Name	Value
CONFIG_SYSTEM_TICK_FREQUENCY	100
CONFIG_SYS_CONFIG_STRING	1
CONFIG_SYS_SIMBA_MAIN_STACK_MAX	4096
CONFIG_THRD_CPU_USAGE	1
CONFIG_THRD_ENV	1
CONFIG_THRD_IDLE_STACK_SIZE	384
CONFIG_THRD_SCHEDULED	1
CONFIG_THRD_TERMINATE	1
CONFIG_USB_DEVICE_PID	0x8037
CONFIG_USB_DEVICE_VID	0x2341

Homepage

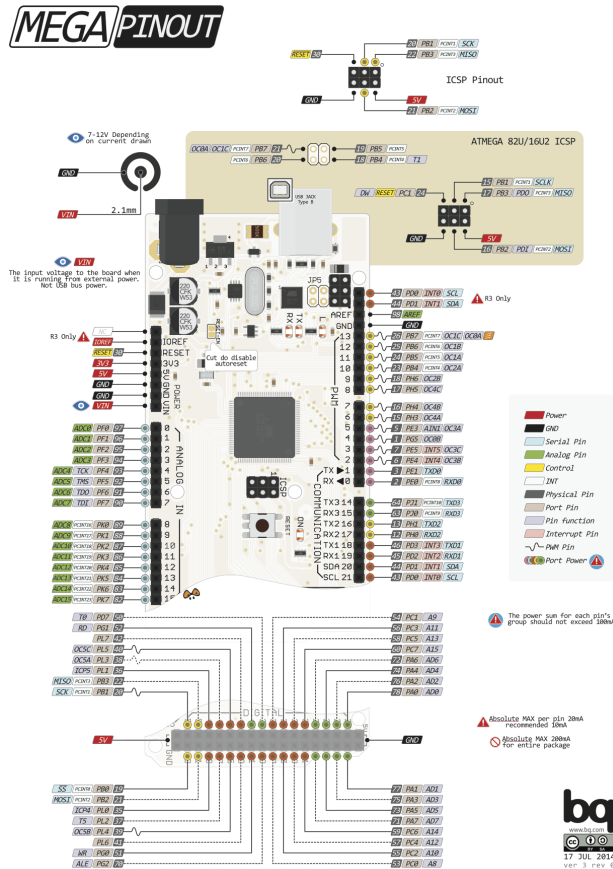
<https://www.arduino.cc/en/Main/ArduinoBoardDue>

Mcu

sam3x8e

1.4.2 Arduino Mega

Pinout



Default system features

The default configuration includes those major features. They are all initialized by `sys_start()` at the startup of the application.

- *Console.*
- *Debug shell.*

Drivers

Supported drivers for this board.

- *adc* — Analog to digital conversion
- *analog_input_pin* — Analog input pin
- *analog_output_pin* — Analog output pin
- *ds18b20* — One-wire temperature sensor
- *ds3231* — RTC clock

- *exti* — External interrupts
- *i2c* — I2C
- *i2c_soft* — Software I2C
- *mcp2515* — CAN BUS chipset
- *nrf24l01* — Wireless communication
- *owi* — One-Wire Interface
- *pin* — Digital pins
- *pwm* — Pulse width modulation
- *sd* — Secure Digital memory
- *spi* — Serial Peripheral Interface
- *uart* — Universal Asynchronous Receiver/Transmitter
- *uart_soft* — Bitbang UART
- *watchdog* — Hardware watchdog

Library Reference

Read more about board specific functionality in the *Arduino Mega* module documentation in the Library Reference.

Memory usage

Below is the memory usage of two applications:

- The [minimal-configuration](#) application is configured to only include the bare minimum of functionality for the low level kernel to run. That is, the thread scheduler and system tick.
- The [default-configuration](#) application is built with the default configuration, including a lot more functionality. See the list of [Default system features](#) above for a summary.

Application	Flash	RAM
minimal-configuration	13674	1053
default-configuration	58156	3737

Default configuration

Default Standard Library configuration.

Name	Value
CONFIG_ASSERT	1
CONFIG_DEBUG	1
CONFIG_FS_CMD_DS18B20_LIST	1
CONFIG_FS_CMD_ESP_WIFI_STATUS	0
CONFIG_FS_CMD_FS_APPEND	1
CONFIG_FS_CMD_FS_COUNTERS_LIST	1
CONFIG_FS_CMD_FS_COUNTERS_RESET	1
CONFIG_FS_CMD_FS_FILESYSTEMS_LIST	1
CONFIG_FS_CMD_FS_FORMAT	1

Continued on next page

Table 1.2 – continued from previous page

Name	Value
CONFIG_FS_CMD_FS_LIST	1
CONFIG_FS_CMD_FS_PARAMETERS_LIST	1
CONFIG_FS_CMD_FS_READ	1
CONFIG_FS_CMD_FS_REMOVE	1
CONFIG_FS_CMD_FS_WRITE	1
CONFIG_FS_CMD_I2C_READ	1
CONFIG_FS_CMD_I2C_WRITE	1
CONFIG_FS_CMD_LOG_LIST	1
CONFIG_FS_CMD_LOG_PRINT	1
CONFIG_FS_CMD_LOG_SET_LOG_MASK	1
CONFIG_FS_CMD_NETWORK_INTERFACE_LIST	1
CONFIG_FS_CMD_PING_PING	1
CONFIG_FS_CMD_PIN_READ	1
CONFIG_FS_CMD_PIN_SET_MODE	1
CONFIG_FS_CMD_PIN_WRITE	1
CONFIG_FS_CMD_SERVICE_LIST	1
CONFIG_FS_CMD_SERVICE_START	1
CONFIG_FS_CMD_SERVICE_STOP	1
CONFIG_FS_CMD_SETTINGS_LIST	1
CONFIG_FS_CMD_SETTINGS_READ	1
CONFIG_FS_CMD_SETTINGS_RESET	1
CONFIG_FS_CMD_SETTINGS_WRITE	1
CONFIG_FS_CMD_SYS_CONFIG	1
CONFIG_FS_CMD_SYS_INFO	1
CONFIG_FS_CMD_SYS_REBOOT	1
CONFIG_FS_CMD_SYS_UPTIME	1
CONFIG_FS_CMD_THRD_LIST	1
CONFIG_FS_CMD_THRD_SET_LOG_MASK	1
CONFIG_FS_CMD_USB_DEVICE_LIST	1
CONFIG_FS_CMD_USB_HOST_LIST	1
CONFIG_FS_PATH_MAX	64
CONFIG_MONITOR_THREAD	1
CONFIG_PREEMPTIVE_SCHEDULER	0
CONFIG_PROFILE_STACK	1
CONFIG_SETTINGS_AREA_SIZE	256
CONFIG_SHELL_COMMAND_MAX	64
CONFIG_SHELL_HISTORY_SIZE	768
CONFIG_SHELL_MINIMAL	0
CONFIG_SHELL_PROMPT	“\$ “
CONFIG_SOCKET_RAW	1
CONFIG_SPIFFS	0
CONFIG_START_CONSOLE	CONFIG_START_CONSOLE_UART
CONFIG_START_CONSOLE_DEVICE_INDEX	0
CONFIG_START_CONSOLE_UART_BAUDRATE	38400
CONFIG_START_CONSOLE_USB_CDC_CONTROL_INTERFACE	0
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_IN	2
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_OUT	3
CONFIG_START_CONSOLE_USB_CDC_WAIT_FOR_CONNETION	1
CONFIG_START_FILESYSTEM	0

Continued on next page

Table 1.2 – continued from previous page

Name	Value
CONFIG_START_FILESYSTEM_ADDRESS	0
CONFIG_START_FILESYSTEM_SIZE	65536
CONFIG_START_NETWORK	0
CONFIG_START_NETWORK_INTERFACE_WIFI_CONNECT_TIMEOUT	30
CONFIG_START_NETWORK_INTERFACE_WIFI_PASSWORD	MyWiFiPassword
CONFIG_START_NETWORK_INTERFACE_WIFI_SSID	MyWiFiSSID
CONFIG_START_SHELL	1
CONFIG_START_SHELL_PRIO	30
CONFIG_START_SHELL_STACK_SIZE	768
CONFIG_STD_OUTPUT_BUFFER_MAX	16
CONFIG_SYSTEM_TICK_FREQUENCY	100
CONFIG_SYS_CONFIG_STRING	1
CONFIG_SYS_SIMBA_MAIN_STACK_MAX	4096
CONFIG_THRD_CPU_USAGE	1
CONFIG_THRD_ENV	1
CONFIG_THRD_IDLE_STACK_SIZE	156
CONFIG_THRD_SCHEDULED	1
CONFIG_THRD_TERMINATE	1
CONFIG_USB_DEVICE_PID	0x8037
CONFIG_USB_DEVICE_VID	0x2341

Homepage

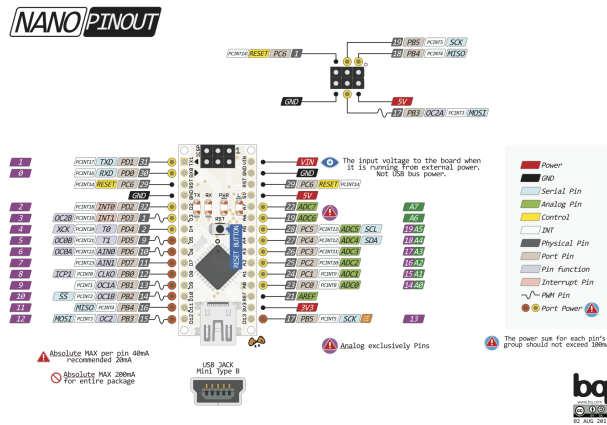
<https://www.arduino.cc/en/Main/ArduinoBoardMega>

Mcu

atmega2560

1.4.3 Arduino Nano

Pinout



Default system features

The default configuration includes those major features. They are all initialized by `sys_start()` at the startup of the application.

- *Console.*

Drivers

Supported drivers for this board.

- *adc — Analog to digital conversion*
- *analog_input_pin — Analog input pin*
- *analog_output_pin — Analog output pin*
- *ds18b20 — One-wire temperature sensor*
- *ds3231 — RTC clock*
- *exti — External interrupts*
- *i2c — I2C*
- *i2c_soft — Software I2C*
- *mcp2515 — CAN BUS chipset*
- *nrf24l01 — Wireless communication*
- *owi — One-Wire Interface*
- *pin — Digital pins*
- *pwm — Pulse width modulation*
- *sd — Secure Digital memory*
- *spi — Serial Peripheral Interface*
- *uart — Universal Asynchronous Receiver/Transmitter*
- *uart_soft — Bitbang UART*
- *watchdog — Hardware watchdog*

Library Reference

Read more about board specific functionality in the [Arduino Nano](#) module documentation in the Library Reference.

Memory usage

Below is the memory usage of two applications:

- The [minimal-configuration](#) application is configured to only include the bare minimum of functionality for the low level kernel to run. That is, the thread scheduler and system tick.
- The [default-configuration](#) application is built with the default configuration, including a lot more functionality. See the list of [Default system features](#) above for a summary.

Application	Flash	RAM
minimal-configuration	5604	755
default-configuration	10998	868

Default configuration

Default Standard Library configuration.

Name	Value
CONFIG_ASSERT	0
CONFIG_DEBUG	1
CONFIG_FS_CMD_DS18B20_LIST	0
CONFIG_FS_CMD_ESP_WIFI_STATUS	0
CONFIG_FS_CMD_FS_APPEND	0
CONFIG_FS_CMD_FS_COUNTERS_LIST	0
CONFIG_FS_CMD_FS_COUNTERS_RESET	0
CONFIG_FS_CMD_FS_FILESYSTEMS_LIST	0
CONFIG_FS_CMD_FS_FORMAT	0
CONFIG_FS_CMD_FS_LIST	0
CONFIG_FS_CMD_FS_PARAMETERS_LIST	0
CONFIG_FS_CMD_FS_READ	0
CONFIG_FS_CMD_FS_REMOVE	0
CONFIG_FS_CMD_FS_WRITE	0
CONFIG_FS_CMD_I2C_READ	0
CONFIG_FS_CMD_I2C_WRITE	0
CONFIG_FS_CMD_LOG_LIST	0
CONFIG_FS_CMD_LOG_PRINT	0
CONFIG_FS_CMD_LOG_SET_LOG_MASK	0
CONFIG_FS_CMD_NETWORK_INTERFACE_LIST	0
CONFIG_FS_CMD_PING_PING	1
CONFIG_FS_CMD_PIN_READ	0
CONFIG_FS_CMD_PIN_SET_MODE	0
CONFIG_FS_CMD_PIN_WRITE	0
CONFIG_FS_CMD_SERVICE_LIST	1
CONFIG_FS_CMD_SERVICE_START	1
CONFIG_FS_CMD_SERVICE_STOP	1
CONFIG_FS_CMD_SETTINGS_LIST	0
CONFIG_FS_CMD_SETTINGS_READ	0
CONFIG_FS_CMD_SETTINGS_RESET	0
CONFIG_FS_CMD_SETTINGS_WRITE	0
CONFIG_FS_CMD_SYS_CONFIG	0
CONFIG_FS_CMD_SYS_INFO	0
CONFIG_FS_CMD_SYS_REBOOT	0
CONFIG_FS_CMD_SYS_UPTIME	0
CONFIG_FS_CMD_THRD_LIST	0
CONFIG_FS_CMD_THRD_SET_LOG_MASK	0
CONFIG_FS_CMD_USB_DEVICE_LIST	0
CONFIG_FS_CMD_USB_HOST_LIST	0
CONFIG_FS_PATH_MAX	64
CONFIG_MONITOR_THREAD	0
CONFIG_PREEMPTIVE_SCHEDULER	0

Continued on next page

Table 1.3 – continued from previous page

Name	Value
CONFIG_PROFILE_STACK	1
CONFIG_SETTINGS_AREA_SIZE	256
CONFIG_SHELL_COMMAND_MAX	64
CONFIG_SHELL_HISTORY_SIZE	768
CONFIG_SHELL_MINIMAL	1
CONFIG_SHELL_PROMPT	“\$ “
CONFIG_SOCKET_RAW	1
CONFIG_SPIFFS	0
CONFIG_START_CONSOLE	CONFIG_START_CONSOLE_UART
CONFIG_START_CONSOLE_DEVICE_INDEX	0
CONFIG_START_CONSOLE_UART_BAUDRATE	38400
CONFIG_START_CONSOLE_USB_CDC_CONTROL_INTERFACE	0
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_IN	2
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_OUT	3
CONFIG_START_CONSOLE_USB_CDC_WAIT_FOR_CONNETION	1
CONFIG_START_FILESYSTEM	0
CONFIG_START_FILESYSTEM_ADDRESS	0
CONFIG_START_FILESYSTEM_SIZE	65536
CONFIG_START_NETWORK	0
CONFIG_START_NETWORK_INTERFACE_WIFI_CONNECT_TIMEOUT	30
CONFIG_START_NETWORK_INTERFACE_WIFI_PASSWORD	MyWiFiPassword
CONFIG_START_NETWORK_INTERFACE_WIFI_SSID	MyWiFiSSID
CONFIG_START_SHELL	0
CONFIG_START_SHELL_PRIO	30
CONFIG_START_SHELL_STACK_SIZE	768
CONFIG_STD_OUTPUT_BUFFER_MAX	16
CONFIG_SYSTEM_TICK_FREQUENCY	100
CONFIG_SYS_CONFIG_STRING	0
CONFIG_SYS_SIMBA_MAIN_STACK_MAX	4096
CONFIG_THRD_CPU_USAGE	1
CONFIG_THRD_ENV	0
CONFIG_THRD_IDLE_STACK_SIZE	156
CONFIG_THRD_SCHEDULED	1
CONFIG_THRD_TERMINATE	1
CONFIG_USB_DEVICE_PID	0x8037
CONFIG_USB_DEVICE_VID	0x2341

Homepage

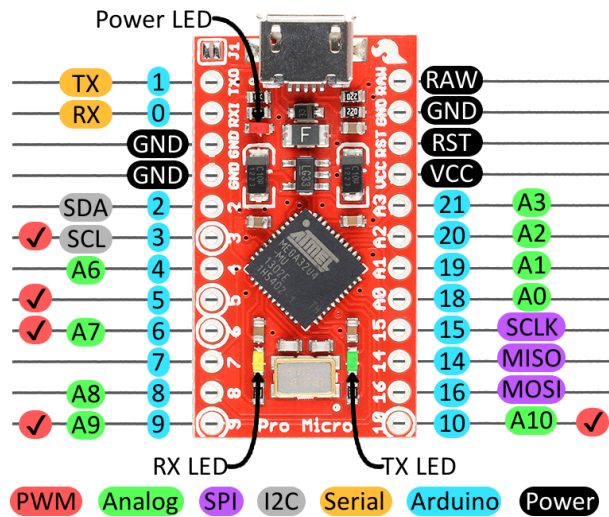
<https://www.arduino.cc/en/Main/ArduinoBoardNano>

Mcu

atmega328p

1.4.4 Arduino Pro Micro

Pinout



Default system features

The default configuration includes those major features. They are all initialized by `sys_start()` at the startup of the application.

- *Console.*

Drivers

Supported drivers for this board.

- *adc* — Analog to digital conversion
- *analog_input_pin* — Analog input pin
- *analog_output_pin* — Analog output pin
- *ds18b20* — One-wire temperature sensor
- *ds3231* — RTC clock
- *exti* — External interrupts
- *i2c* — I2C
- *i2c_soft* — Software I2C
- *mcp2515* — CAN BUS chipset
- *nrf24l01* — Wireless communication
- *owi* — One-Wire Interface
- *pin* — Digital pins
- *pwm* — Pulse width modulation
- *sd* — Secure Digital memory

- *spi* — *Serial Peripheral Interface*
- *uart* — *Universal Asynchronous Receiver/Transmitter*
- *uart_soft* — *Bitbang UART*
- *usb* — *Universal Serial Bus*
- *usb_device* — *Universal Serial Bus - Device*
- *watchdog* — *Hardware watchdog*

Library Reference

Read more about board specific functionality in the *Arduino Pro Micro* module documentation in the Library Reference.

Memory usage

Below is the memory usage of two applications:

- The *minimal-configuration* application is configured to only include the bare minimum of functionality for the low level kernel to run. That is, the thread scheduler and system tick.
- The *default-configuration* application is built with the default configuration, including a lot more functionality. See the list of *Default system features* above for a summary.

Application	Flash	RAM
minimal-configuration	6562	837
default-configuration	10078	993

Default configuration

Default Standard Library configuration.

Name	Value
CONFIG_ASSERT	0
CONFIG_DEBUG	1
CONFIG_FS_CMD_DS18B20_LIST	0
CONFIG_FS_CMD_ESP_WIFI_STATUS	0
CONFIG_FS_CMD_FS_APPEND	0
CONFIG_FS_CMD_FS_COUNTERS_LIST	0
CONFIG_FS_CMD_FS_COUNTERS_RESET	0
CONFIG_FS_CMD_FS_FILESYSTEMS_LIST	0
CONFIG_FS_CMD_FS_FORMAT	0
CONFIG_FS_CMD_FS_LIST	0
CONFIG_FS_CMD_FS_PARAMETERS_LIST	0
CONFIG_FS_CMD_FS_READ	0
CONFIG_FS_CMD_FS_REMOVE	0
CONFIG_FS_CMD_FS_WRITE	0
CONFIG_FS_CMD_I2C_READ	0
CONFIG_FS_CMD_I2C_WRITE	0
CONFIG_FS_CMD_LOG_LIST	0
CONFIG_FS_CMD_LOG_PRINT	0

Continued on next page

Table 1.4 – continued from previous page

Name	Value
CONFIG_FS_CMD_LOG_SET_LOG_MASK	0
CONFIG_FS_CMD_NETWORK_INTERFACE_LIST	0
CONFIG_FS_CMD_PING_PING	1
CONFIG_FS_CMD_PIN_READ	0
CONFIG_FS_CMD_PIN_SET_MODE	0
CONFIG_FS_CMD_PIN_WRITE	0
CONFIG_FS_CMD_SERVICE_LIST	1
CONFIG_FS_CMD_SERVICE_START	1
CONFIG_FS_CMD_SERVICE_STOP	1
CONFIG_FS_CMD_SETTINGS_LIST	0
CONFIG_FS_CMD_SETTINGS_READ	0
CONFIG_FS_CMD_SETTINGS_RESET	0
CONFIG_FS_CMD_SETTINGS_WRITE	0
CONFIG_FS_CMD_SYS_CONFIG	0
CONFIG_FS_CMD_SYS_INFO	0
CONFIG_FS_CMD_SYS_REBOOT	0
CONFIG_FS_CMD_SYS_UPTIME	0
CONFIG_FS_CMD_THRD_LIST	0
CONFIG_FS_CMD_THRD_SET_LOG_MASK	0
CONFIG_FS_CMD_USB_DEVICE_LIST	0
CONFIG_FS_CMD_USB_HOST_LIST	0
CONFIG_FS_PATH_MAX	64
CONFIG_MONITOR_THREAD	0
CONFIG_PREEMPTIVE_SCHEDULER	0
CONFIG_PROFILE_STACK	1
CONFIG_SETTINGS_AREA_SIZE	256
CONFIG_SHELL_COMMAND_MAX	64
CONFIG_SHELL_HISTORY_SIZE	768
CONFIG_SHELL_MINIMAL	1
CONFIG_SHELL_PROMPT	“\$ “
CONFIG_SOCKET_RAW	1
CONFIG_SPIFFS	0
CONFIG_START_CONSOLE	CONFIG_START_CONSOLE_USB_CDC
CONFIG_START_CONSOLE_DEVICE_INDEX	0
CONFIG_START_CONSOLE_UART_BAUDRATE	38400
CONFIG_START_CONSOLE_USB_CDC_CONTROL_INTERFACE	0
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_IN	2
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_OUT	3
CONFIG_START_CONSOLE_USB_CDC_WAIT_FOR_CONNCTION	1
CONFIG_START_FILESYSTEM	0
CONFIG_START_FILESYSTEM_ADDRESS	0
CONFIG_START_FILESYSTEM_SIZE	65536
CONFIG_START_NETWORK	0
CONFIG_START_NETWORK_INTERFACE_WIFI_CONNECT_TIMEOUT	30
CONFIG_START_NETWORK_INTERFACE_WIFI_PASSWORD	MyWiFiPassword
CONFIG_START_NETWORK_INTERFACE_WIFI_SSID	MyWiFiSSID
CONFIG_START_SHELL	0
CONFIG_START_SHELL_PRIO	30
CONFIG_START_SHELL_STACK_SIZE	768

Continued on next page

Table 1.4 – continued from previous page

Name	Value
CONFIG_STD_OUTPUT_BUFFER_MAX	16
CONFIG_SYSTEM_TICK_FREQUENCY	100
CONFIG_SYS_CONFIG_STRING	0
CONFIG_SYS_SIMBA_MAIN_STACK_MAX	4096
CONFIG_THRD_CPU_USAGE	1
CONFIG_THRD_ENV	0
CONFIG_THRD_IDLE_STACK_SIZE	156
CONFIG_THRD_SCHEDULED	1
CONFIG_THRD_TERMINATE	1
CONFIG_USB_DEVICE_PID	0x8037
CONFIG_USB_DEVICE_VID	0x2341

Homepage

<https://www.sparkfun.com/products/12640>

Mcu

atmega32u4

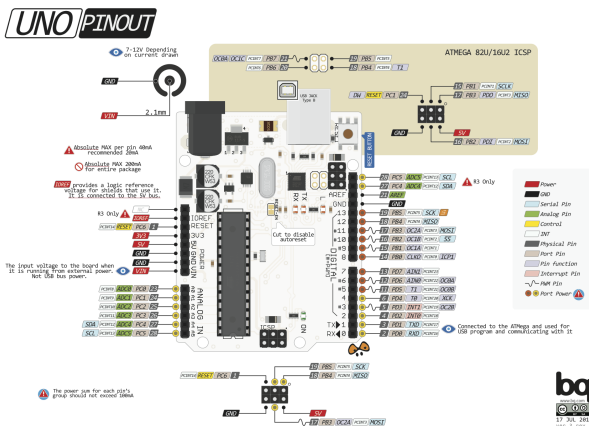
Enter the bootloader

Recover a bricked board by entering the bootloader.

1. Power up the board.
2. Connect RST to GND for a second to enter the bootloader and stay in it for 8 seconds.

1.4.5 Arduino Uno

Pinout



Default system features

The default configuration includes those major features. They are all initialized by `sys_start()` at the startup of the application.

- *Console.*

Drivers

Supported drivers for this board.

- *adc — Analog to digital conversion*
- *analog_input_pin — Analog input pin*
- *analog_output_pin — Analog output pin*
- *ds18b20 — One-wire temperature sensor*
- *ds3231 — RTC clock*
- *exti — External interrupts*
- *i2c — I2C*
- *i2c_soft — Software I2C*
- *mcp2515 — CAN BUS chipset*
- *nrf24l01 — Wireless communication*
- *owi — One-Wire Interface*
- *pin — Digital pins*
- *pwm — Pulse width modulation*
- *sd — Secure Digital memory*
- *spi — Serial Peripheral Interface*
- *uart — Universal Asynchronous Receiver/Transmitter*
- *uart_soft — Bitbang UART*
- *watchdog — Hardware watchdog*

Library Reference

Read more about board specific functionality in the [Arduino Uno](#) module documentation in the Library Reference.

Memory usage

Below is the memory usage of two applications:

- The [minimal-configuration](#) application is configured to only include the bare minimum of functionality for the low level kernel to run. That is, the thread scheduler and system tick.
- The [default-configuration](#) application is built with the default configuration, including a lot more functionality. See the list of [Default system features](#) above for a summary.

Application	Flash	RAM
minimal-configuration	5604	755
default-configuration	10998	868

Default configuration

Default Standard Library configuration.

Name	Value
CONFIG_ASSERT	0
CONFIG_DEBUG	1
CONFIG_FS_CMD_DS18B20_LIST	0
CONFIG_FS_CMD_ESP_WIFI_STATUS	0
CONFIG_FS_CMD_FS_APPEND	0
CONFIG_FS_CMD_FS_COUNTERS_LIST	0
CONFIG_FS_CMD_FS_COUNTERS_RESET	0
CONFIG_FS_CMD_FS_FILESYSTEMS_LIST	0
CONFIG_FS_CMD_FS_FORMAT	0
CONFIG_FS_CMD_FS_LIST	0
CONFIG_FS_CMD_FS_PARAMETERS_LIST	0
CONFIG_FS_CMD_FS_READ	0
CONFIG_FS_CMD_FS_REMOVE	0
CONFIG_FS_CMD_FS_WRITE	0
CONFIG_FS_CMD_I2C_READ	0
CONFIG_FS_CMD_I2C_WRITE	0
CONFIG_FS_CMD_LOG_LIST	0
CONFIG_FS_CMD_LOG_PRINT	0
CONFIG_FS_CMD_LOG_SET_LOG_MASK	0
CONFIG_FS_CMD_NETWORK_INTERFACE_LIST	0
CONFIG_FS_CMD_PING_PING	1
CONFIG_FS_CMD_PIN_READ	0
CONFIG_FS_CMD_PIN_SET_MODE	0
CONFIG_FS_CMD_PIN_WRITE	0
CONFIG_FS_CMD_SERVICE_LIST	1
CONFIG_FS_CMD_SERVICE_START	1
CONFIG_FS_CMD_SERVICE_STOP	1
CONFIG_FS_CMD_SETTINGS_LIST	0
CONFIG_FS_CMD_SETTINGS_READ	0
CONFIG_FS_CMD_SETTINGS_RESET	0
CONFIG_FS_CMD_SETTINGS_WRITE	0
CONFIG_FS_CMD_SYS_CONFIG	0
CONFIG_FS_CMD_SYS_INFO	0
CONFIG_FS_CMD_SYS_REBOOT	0
CONFIG_FS_CMD_SYS_UPTIME	0
CONFIG_FS_CMD_THRD_LIST	0
CONFIG_FS_CMD_THRD_SET_LOG_MASK	0
CONFIG_FS_CMD_USB_DEVICE_LIST	0
CONFIG_FS_CMD_USB_HOST_LIST	0
CONFIG_FS_PATH_MAX	64
CONFIG_MONITOR_THREAD	0
CONFIG_PREEMPTIVE_SCHEDULER	0
Continued on next page	

Table 1.5 – continued from previous page

Name	Value
CONFIG_PROFILE_STACK	1
CONFIG_SETTINGS_AREA_SIZE	256
CONFIG_SHELL_COMMAND_MAX	64
CONFIG_SHELL_HISTORY_SIZE	768
CONFIG_SHELL_MINIMAL	1
CONFIG_SHELL_PROMPT	“\$ “
CONFIG_SOCKET_RAW	1
CONFIG_SPIFFS	0
CONFIG_START_CONSOLE	CONFIG_START_CONSOLE_UART
CONFIG_START_CONSOLE_DEVICE_INDEX	0
CONFIG_START_CONSOLE_UART_BAUDRATE	38400
CONFIG_START_CONSOLE_USB_CDC_CONTROL_INTERFACE	0
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_IN	2
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_OUT	3
CONFIG_START_CONSOLE_USB_CDC_WAIT_FOR_CONNETION	1
CONFIG_START_FILESYSTEM	0
CONFIG_START_FILESYSTEM_ADDRESS	0
CONFIG_START_FILESYSTEM_SIZE	65536
CONFIG_START_NETWORK	0
CONFIG_START_NETWORK_INTERFACE_WIFI_CONNECT_TIMEOUT	30
CONFIG_START_NETWORK_INTERFACE_WIFI_PASSWORD	MyWiFiPassword
CONFIG_START_NETWORK_INTERFACE_WIFI_SSID	MyWiFiSSID
CONFIG_START_SHELL	0
CONFIG_START_SHELL_PRIO	30
CONFIG_START_SHELL_STACK_SIZE	768
CONFIG_STD_OUTPUT_BUFFER_MAX	16
CONFIG_SYSTEM_TICK_FREQUENCY	100
CONFIG_SYS_CONFIG_STRING	0
CONFIG_SYS_SIMBA_MAIN_STACK_MAX	4096
CONFIG_THRD_CPU_USAGE	1
CONFIG_THRD_ENV	0
CONFIG_THRD_IDLE_STACK_SIZE	156
CONFIG_THRD_SCHEDULED	1
CONFIG_THRD_TERMINATE	1
CONFIG_USB_DEVICE_PID	0x8037
CONFIG_USB_DEVICE_VID	0x2341

Homepage

<https://www.arduino.cc/en/Main/ArduinoBoardUno>

Mcu

atmega328p

1.4.6 Cygwin

Pinout



Default system features

The default configuration includes those major features. They are all initialized by `sys_start()` at the startup of the application.

- *Console.*
- *File system.*
- *Debug shell.*

Drivers

Supported drivers for this board.

- *adc — Analog to digital conversion*
- *analog_input_pin — Analog input pin*
- *analog_output_pin — Analog output pin*
- *can — CAN bus*
- *dac — Digital to analog conversion*
- *ds18b20 — One-wire temperature sensor*
- *exti — External interrupts*
- *flash — Flash memory*
- *i2c_soft — Software I2C*
- *owi — One-Wire Interface*

- *pin* — Digital pins
- *pwm* — Pulse width modulation
- *sd* — Secure Digital memory
- *spi* — Serial Peripheral Interface
- *uart* — Universal Asynchronous Receiver/Transmitter

Library Reference

Read more about board specific functionality in the *Cygwin* module documentation in the Library Reference.

Memory usage

Below is the memory usage of two applications:

- The *minimal-configuration* application is configured to only include the bare minimum of functionality for the low level kernel to run. That is, the thread scheduler and system tick.
- The *default-configuration* application is built with the default configuration, including a lot more functionality. See the list of *Default system features* above for a summary.

Application	Flash	RAM
minimal-configuration	312535	120632
default-configuration	403367	211896

Default configuration

Default Standard Library configuration.

Name	Value
CONFIG_ASSERT	1
CONFIG_DEBUG	1
CONFIG_FS_CMD_DS18B20_LIST	1
CONFIG_FS_CMD_ESP_WIFI_STATUS	0
CONFIG_FS_CMD_FS_APPEND	1
CONFIG_FS_CMD_FS_COUNTERS_LIST	1
CONFIG_FS_CMD_FS_COUNTERS_RESET	1
CONFIG_FS_CMD_FS_FILESYSTEMS_LIST	1
CONFIG_FS_CMD_FS_FORMAT	1
CONFIG_FS_CMD_FS_LIST	1
CONFIG_FS_CMD_FS_PARAMETERS_LIST	1
CONFIG_FS_CMD_FS_READ	1
CONFIG_FS_CMD_FS_REMOVE	1
CONFIG_FS_CMD_FS_WRITE	1
CONFIG_FS_CMD_I2C_READ	1
CONFIG_FS_CMD_I2C_WRITE	1
CONFIG_FS_CMD_LOG_LIST	1
CONFIG_FS_CMD_LOG_PRINT	1
CONFIG_FS_CMD_LOG_SET_LOG_MASK	1
CONFIG_FS_CMD_NETWORK_INTERFACE_LIST	1
Continued on next page	

Table 1.6 – continued from previous page

Name	Value
CONFIG_FS_CMD_PING_PING	1
CONFIG_FS_CMD_PIN_READ	1
CONFIG_FS_CMD_PIN_SET_MODE	1
CONFIG_FS_CMD_PIN_WRITE	1
CONFIG_FS_CMD_SERVICE_LIST	1
CONFIG_FS_CMD_SERVICE_START	1
CONFIG_FS_CMD_SERVICE_STOP	1
CONFIG_FS_CMD_SETTINGS_LIST	1
CONFIG_FS_CMD_SETTINGS_READ	1
CONFIG_FS_CMD_SETTINGS_RESET	1
CONFIG_FS_CMD_SETTINGS_WRITE	1
CONFIG_FS_CMD_SYS_CONFIG	1
CONFIG_FS_CMD_SYS_INFO	1
CONFIG_FS_CMD_SYS_REBOOT	1
CONFIG_FS_CMD_SYS_UPTIME	1
CONFIG_FS_CMD_THRD_LIST	1
CONFIG_FS_CMD_THRD_SET_LOG_MASK	1
CONFIG_FS_CMD_USB_DEVICE_LIST	1
CONFIG_FS_CMD_USB_HOST_LIST	1
CONFIG_FS_PATH_MAX	64
CONFIG_MONITOR_THREAD	1
CONFIG_PREEMPTIVE_SCHEDULER	0
CONFIG_PROFILE_STACK	1
CONFIG_SETTINGS_AREA_SIZE	256
CONFIG_SHELL_COMMAND_MAX	64
CONFIG_SHELL_HISTORY_SIZE	768
CONFIG_SHELL_MINIMAL	0
CONFIG_SHELL_PROMPT	“\$ “
CONFIG_SOCKET_RAW	1
CONFIG_SPIFFS	1
CONFIG_START_CONSOLE	CONFIG_START_CONSOLE_UART
CONFIG_START_CONSOLE_DEVICE_INDEX	0
CONFIG_START_CONSOLE_UART_BAUDRATE	38400
CONFIG_START_CONSOLE_USB_CDC_CONTROL_INTERFACE	0
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_IN	2
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_OUT	3
CONFIG_START_CONSOLE_USB_CDC_WAIT_FOR_CONNETION	1
CONFIG_START_FILESYSTEM	1
CONFIG_START_FILESYSTEM_ADDRESS	0
CONFIG_START_FILESYSTEM_SIZE	65536
CONFIG_START_NETWORK	0
CONFIG_START_NETWORK_INTERFACE_WIFI_CONNECT_TIMEOUT	30
CONFIG_START_NETWORK_INTERFACE_WIFI_PASSWORD	MyWiFiPassword
CONFIG_START_NETWORK_INTERFACE_WIFI_SSID	MyWiFiSSID
CONFIG_START_SHELL	1
CONFIG_START_SHELL_PRIO	30
CONFIG_START_SHELL_STACK_SIZE	768
CONFIG_STD_OUTPUT_BUFFER_MAX	16
CONFIG_SYSTEM_TICK_FREQUENCY	100

Continued on next page

Table 1.6 – continued from previous page

Name	Value
CONFIG_SYS_CONFIG_STRING	1
CONFIG_SYS_SIMBA_MAIN_STACK_MAX	4096
CONFIG_THRD_CPU_USAGE	1
CONFIG_THRD_ENV	1
CONFIG_THRD_IDLE_STACK_SIZE	1024
CONFIG_THRD_SCHEDULED	1
CONFIG_THRD_TERMINATE	1
CONFIG_USB_DEVICE_PID	0x8037
CONFIG_USB_DEVICE_VID	0x2341

Homepage

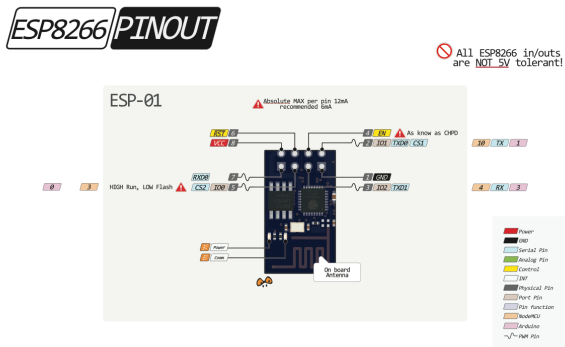
<http://www.cygwin.com>

Mcu

linux

1.4.7 ESP-01

Pinout



Default system features

The default configuration includes those major features. They are all initialized by `sys_start()` at the startup of the application.

- *Console.*
- File system.
- Networking.
- *Debug shell.*

Drivers

Supported drivers for this board.

- *adc* — Analog to digital conversion
- *analog_input_pin* — Analog input pin
- *esp_wifi* — Espressif WiFi
- *exti* — External interrupts
- *flash* — Flash memory
- *i2c_soft* — Software I2C
- *pin* — Digital pins
- *spi* — Serial Peripheral Interface
- *uart* — Universal Asynchronous Receiver/Transmitter
- *uart_soft* — Bitbang UART

Library Reference

Read more about board specific functionality in the [ESP-01](#) module documentation in the Library Reference.

Memory usage

Below is the memory usage of two applications:

- The [minimal-configuration](#) application is configured to only include the bare minimum of functionality for the low level kernel to run. That is, the thread scheduler and system tick.
- The [default-configuration](#) application is built with the default configuration, including a lot more functionality. See the list of [Default system features](#) above for a summary.

Application	Flash	RAM
minimal-configuration	269288	35732
default-configuration	313984	59104

Default configuration

Default Standard Library configuration.

Name	Value
CONFIG_ASSERT	1
CONFIG_DEBUG	1
CONFIG_FS_CMD_DS18B20_LIST	1
CONFIG_FS_CMD_ESP_WIFI_STATUS	1
CONFIG_FS_CMD_FS_APPEND	1
CONFIG_FS_CMD_FS_COUNTERS_LIST	1
CONFIG_FS_CMD_FS_COUNTERS_RESET	1
CONFIG_FS_CMD_FS_FILESYSTEMS_LIST	1
CONFIG_FS_CMD_FS_FORMAT	1
CONFIG_FS_CMD_FS_LIST	1

Continued on next page

Table 1.7 – continued from previous page

Name	Value
CONFIG_FS_CMD_FS_PARAMETERS_LIST	1
CONFIG_FS_CMD_FS_READ	1
CONFIG_FS_CMD_FS_REMOVE	1
CONFIG_FS_CMD_FS_WRITE	1
CONFIG_FS_CMD_I2C_READ	1
CONFIG_FS_CMD_I2C_WRITE	1
CONFIG_FS_CMD_LOG_LIST	1
CONFIG_FS_CMD_LOG_PRINT	1
CONFIG_FS_CMD_LOG_SET_LOG_MASK	1
CONFIG_FS_CMD_NETWORK_INTERFACE_LIST	1
CONFIG_FS_CMD_PING_PING	1
CONFIG_FS_CMD_PIN_READ	1
CONFIG_FS_CMD_PIN_SET_MODE	1
CONFIG_FS_CMD_PIN_WRITE	1
CONFIG_FS_CMD_SERVICE_LIST	1
CONFIG_FS_CMD_SERVICE_START	1
CONFIG_FS_CMD_SERVICE_STOP	1
CONFIG_FS_CMD_SETTINGS_LIST	1
CONFIG_FS_CMD_SETTINGS_READ	1
CONFIG_FS_CMD_SETTINGS_RESET	1
CONFIG_FS_CMD_SETTINGS_WRITE	1
CONFIG_FS_CMD_SYS_CONFIG	1
CONFIG_FS_CMD_SYS_INFO	1
CONFIG_FS_CMD_SYS_REBOOT	1
CONFIG_FS_CMD_SYS_UPTIME	1
CONFIG_FS_CMD_THRD_LIST	1
CONFIG_FS_CMD_THRD_SET_LOG_MASK	1
CONFIG_FS_CMD_USB_DEVICE_LIST	1
CONFIG_FS_CMD_USB_HOST_LIST	1
CONFIG_FS_PATH_MAX	64
CONFIG_MONITOR_THREAD	0
CONFIG_PREEMPTIVE_SCHEDULER	0
CONFIG_PROFILE_STACK	1
CONFIG_SETTINGS_AREA_SIZE	256
CONFIG_SHELL_COMMAND_MAX	64
CONFIG_SHELL_HISTORY_SIZE	768
CONFIG_SHELL_MINIMAL	0
CONFIG_SHELL_PROMPT	“\$ “
CONFIG_SOCKET_RAW	1
CONFIG_SPIFFS	1
CONFIG_START_CONSOLE	CONFIG_START_CONSOLE_UART
CONFIG_START_CONSOLE_DEVICE_INDEX	0
CONFIG_START_CONSOLE_UART_BAUDRATE	76800
CONFIG_START_CONSOLE_USB_CDC_CONTROL_INTERFACE	0
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_IN	2
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_OUT	3
CONFIG_START_CONSOLE_USB_CDC_WAIT_FOR_CONNECTION	1
CONFIG_START_FILESYSTEM	1
CONFIG_START_FILESYSTEM_ADDRESS	0x0006b000

Continued on next page

Table 1.7 – continued from previous page

Name	Value
CONFIG_START_FILESYSTEM_SIZE	0x10000
CONFIG_START_NETWORK	1
CONFIG_START_NETWORK_INTERFACE_WIFI_CONNECT_TIMEOUT	30
CONFIG_START_NETWORK_INTERFACE_WIFI_PASSWORD	MyWiFiPassword
CONFIG_START_NETWORK_INTERFACE_WIFI_SSID	MyWiFiSSID
CONFIG_START_SHELL	1
CONFIG_START_SHELL_PRIO	30
CONFIG_START_SHELL_STACK_SIZE	1536
CONFIG_STD_OUTPUT_BUFFER_MAX	16
CONFIG_SYSTEM_TICK_FREQUENCY	100
CONFIG_SYS_CONFIG_STRING	1
CONFIG_SYS_SIMBA_MAIN_STACK_MAX	4096
CONFIG_THRD_CPU_USAGE	1
CONFIG_THRD_ENV	1
CONFIG_THRD_IDLE_STACK_SIZE	768
CONFIG_THRD_SCHEDULED	1
CONFIG_THRD_TERMINATE	1
CONFIG_USB_DEVICE_PID	0x8037
CONFIG_USB_DEVICE_VID	0x2341

Homepage

<http://espressif.com>

Mcu

esp8266

Flashing

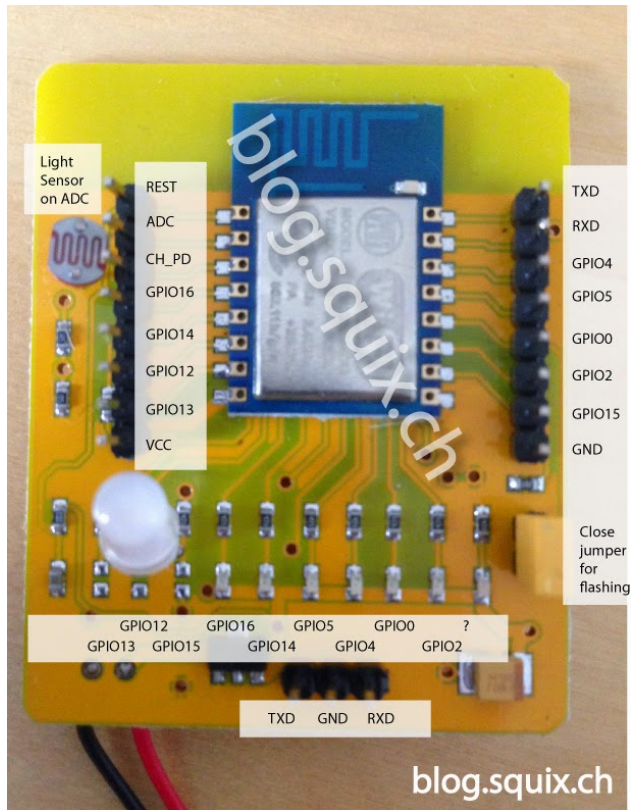
1. Connect VCC to 3.3 V and GND to ground.
2. Connect GPIO0 to GND.
3. Connect EN/CHPH to 3.3 V.
4. Turn on the power.
5. Upload the software to Flash using esptool.

Boot from flash

1. Connect VCC to 3.3 V and GND to ground.
2. Connect GPIO0 to 3.3 V.
3. Connect EN/CHPH to 3.3 V.
4. Turn on the power.

1.4.8 ESP-12E Development Board

Pinout



Default system features

The default configuration includes those major features. They are all initialized by `sys_start()` at the startup of the application.

- *Console.*
- File system.
- Networking.
- *Debug shell.*

Drivers

Supported drivers for this board.

- *adc* — Analog to digital conversion
- *analog_input_pin* — Analog input pin
- *esp_wifi* — Espressif WiFi
- *exti* — External interrupts
- *flash* — Flash memory

- *i2c_soft* — *Software I2C*
- *pin* — *Digital pins*
- *spi* — *Serial Peripheral Interface*
- *uart* — *Universal Asynchronous Receiver/Transmitter*
- *uart_soft* — *Bitbang UART*

Library Reference

Read more about board specific functionality in the *ESP-12E Development Board* module documentation in the Library Reference.

Memory usage

Below is the memory usage of two applications:

- The *minimal-configuration* application is configured to only include the bare minimum of functionality for the low level kernel to run. That is, the thread scheduler and system tick.
- The *default-configuration* application is built with the default configuration, including a lot more functionality. See the list of *Default system features* above for a summary.

Application	Flash	RAM
minimal-configuration	269288	35732
default-configuration	313984	59132

Default configuration

Default Standard Library configuration.

Name	Value
CONFIG_ASSERT	1
CONFIG_DEBUG	1
CONFIG_FS_CMD_DS18B20_LIST	1
CONFIG_FS_CMD_ESP_WIFI_STATUS	1
CONFIG_FS_CMD_FS_APPEND	1
CONFIG_FS_CMD_FS_COUNTERS_LIST	1
CONFIG_FS_CMD_FS_COUNTERS_RESET	1
CONFIG_FS_CMD_FS_FILESYSTEMS_LIST	1
CONFIG_FS_CMD_FS_FORMAT	1
CONFIG_FS_CMD_FS_LIST	1
CONFIG_FS_CMD_FS_PARAMETERS_LIST	1
CONFIG_FS_CMD_FS_READ	1
CONFIG_FS_CMD_FS_REMOVE	1
CONFIG_FS_CMD_FS_WRITE	1
CONFIG_FS_CMD_I2C_READ	1
CONFIG_FS_CMD_I2C_WRITE	1
CONFIG_FS_CMD_LOG_LIST	1
CONFIG_FS_CMD_LOG_PRINT	1
CONFIG_FS_CMD_LOG_SET_LOG_MASK	1
Continued on next page	

Table 1.8 – continued from previous page

Name	Value
CONFIG_FS_CMD_NETWORK_INTERFACE_LIST	1
CONFIG_FS_CMD_PING_PING	1
CONFIG_FS_CMD_PIN_READ	1
CONFIG_FS_CMD_PIN_SET_MODE	1
CONFIG_FS_CMD_PIN_WRITE	1
CONFIG_FS_CMD_SERVICE_LIST	1
CONFIG_FS_CMD_SERVICE_START	1
CONFIG_FS_CMD_SERVICE_STOP	1
CONFIG_FS_CMD_SETTINGS_LIST	1
CONFIG_FS_CMD_SETTINGS_READ	1
CONFIG_FS_CMD_SETTINGS_RESET	1
CONFIG_FS_CMD_SETTINGS_WRITE	1
CONFIG_FS_CMD_SYS_CONFIG	1
CONFIG_FS_CMD_SYS_INFO	1
CONFIG_FS_CMD_SYS_REBOOT	1
CONFIG_FS_CMD_SYS_UPTIME	1
CONFIG_FS_CMD_THRD_LIST	1
CONFIG_FS_CMD_THRD_SET_LOG_MASK	1
CONFIG_FS_CMD_USB_DEVICE_LIST	1
CONFIG_FS_CMD_USB_HOST_LIST	1
CONFIG_FS_PATH_MAX	64
CONFIG_MONITOR_THREAD	0
CONFIG_PREEMPTIVE_SCHEDULER	0
CONFIG_PROFILE_STACK	1
CONFIG_SETTINGS_AREA_SIZE	256
CONFIG_SHELL_COMMAND_MAX	64
CONFIG_SHELL_HISTORY_SIZE	768
CONFIG_SHELL_MINIMAL	0
CONFIG_SHELL_PROMPT	“\$ “
CONFIG_SOCKET_RAW	1
CONFIG_SPIFFS	1
CONFIG_START_CONSOLE	CONFIG_START_CONSOLE_UART
CONFIG_START_CONSOLE_DEVICE_INDEX	0
CONFIG_START_CONSOLE_UART_BAUDRATE	76800
CONFIG_START_CONSOLE_USB_CDC_CONTROL_INTERFACE	0
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_IN	2
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_OUT	3
CONFIG_START_CONSOLE_USB_CDC_WAIT_FOR_CONNETION	1
CONFIG_START_FILESYSTEM	1
CONFIG_START_FILESYSTEM_ADDRESS	0x00300000
CONFIG_START_FILESYSTEM_SIZE	0xFB000
CONFIG_START_NETWORK	1
CONFIG_START_NETWORK_INTERFACE_WIFI_CONNECT_TIMEOUT	30
CONFIG_START_NETWORK_INTERFACE_WIFI_PASSWORD	MyWiFiPassword
CONFIG_START_NETWORK_INTERFACE_WIFI_SSID	MyWiFiSSID
CONFIG_START_SHELL	1
CONFIG_START_SHELL_PRIO	30
CONFIG_START_SHELL_STACK_SIZE	1536
CONFIG_STD_OUTPUT_BUFFER_MAX	16

Continued on next page

Table 1.8 – continued from previous page

Name	Value
CONFIG_SYSTEM_TICK_FREQUENCY	100
CONFIG_SYS_CONFIG_STRING	1
CONFIG_SYS_SIMBA_MAIN_STACK_MAX	4096
CONFIG_THRD_CPU_USAGE	1
CONFIG_THRD_ENV	1
CONFIG_THRD_IDLE_STACK_SIZE	768
CONFIG_THRD_SCHEDULED	1
CONFIG_THRD_TERMINATE	1
CONFIG_USB_DEVICE_PID	0x8037
CONFIG_USB_DEVICE_VID	0x2341

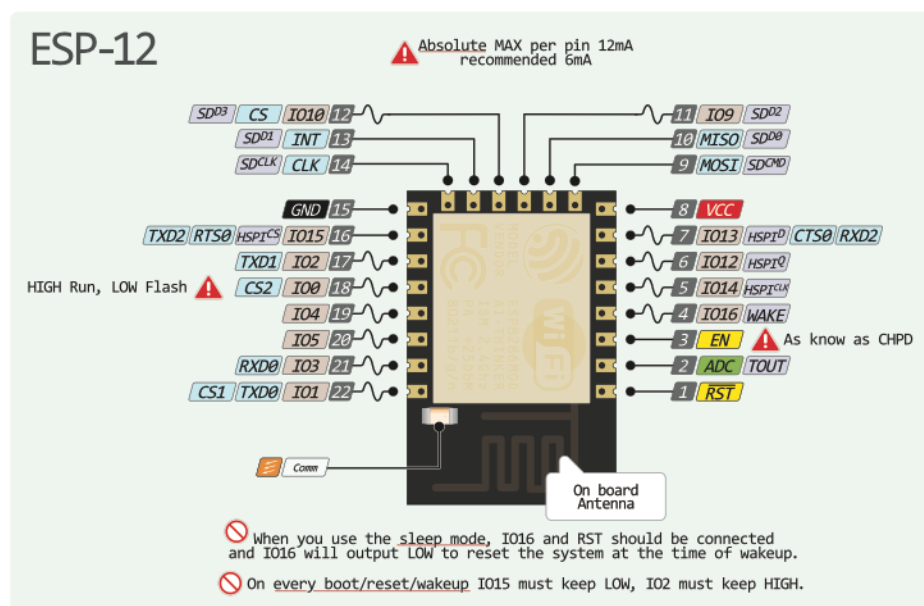
Homepage

<http://espressif.com>

Mcu

esp8266

ESP-12 pinout



Flashing

1. Connect 3.3 V to VCC and ground to GND.
2. Attach the flash jumper (to the right in the picture).
3. Turn on the power.
4. Upload the software to Flash using esptool.

5. The application starts automatically when the download is completed.

Hardware

- 3.3 V power supply and logical level voltage.
- Boot message at 76800 baud on a virgin board. Blue, red and RGB LEDs turned on.
- 4 MB Flash.

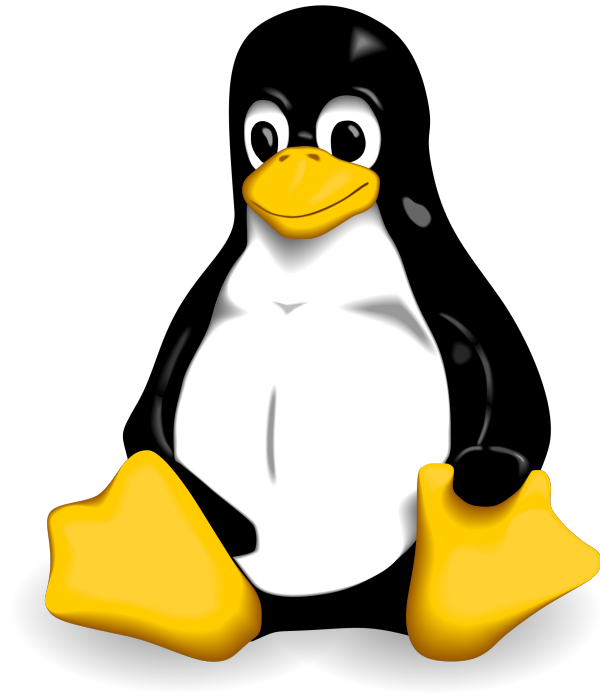
How to determine the Flash size:

```
$ python esptool.py --port /dev/ttyUSB0 flash_id
Connecting...
head: 0 ;total: 0
erase size : 0
Manufacturer: e0
Device: 4016
```

Device 4016 gives a Flash of size $2^{(16-1)} / 8 = 4096 \text{ kB} = 4 \text{ MB}$.

1.4.9 Linux

Pinout



Default system features

The default configuration includes those major features. They are all initialized by `sys_start()` at the startup of the application.

- *Console.*
- File system.
- *Debug shell.*

Drivers

Supported drivers for this board.

- *adc* — Analog to digital conversion
- *analog_input_pin* — Analog input pin
- *analog_output_pin* — Analog output pin
- *can* — CAN bus
- *dac* — Digital to analog conversion
- *ds18b20* — One-wire temperature sensor
- *exti* — External interrupts
- *flash* — Flash memory
- *i2c_soft* — Software I2C
- *owi* — One-Wire Interface
- *pin* — Digital pins
- *pwm* — Pulse width modulation
- *sd* — Secure Digital memory
- *spi* — Serial Peripheral Interface
- *uart* — Universal Asynchronous Receiver/Transmitter

Library Reference

Read more about board specific functionality in the [Linux](#) module documentation in the Library Reference.

Memory usage

Below is the memory usage of two applications:

- The [minimal-configuration](#) application is configured to only include the bare minimum of functionality for the low level kernel to run. That is, the thread scheduler and system tick.
- The [default-configuration](#) application is built with the default configuration, including a lot more functionality. See the list of [Default system features](#) above for a summary.

Application	Flash	RAM
minimal-configuration	312439	120632
default-configuration	403303	211896

Default configuration

Default Standard Library configuration.

Name	Value
CONFIG_ASSERT	1
CONFIG_DEBUG	1
CONFIG_FS_CMD_DS18B20_LIST	1
CONFIG_FS_CMD_ESP_WIFI_STATUS	0
CONFIG_FS_CMD_FS_APPEND	1
CONFIG_FS_CMD_FS_COUNTERS_LIST	1
CONFIG_FS_CMD_FS_COUNTERS_RESET	1
CONFIG_FS_CMD_FS_FILESYSTEMS_LIST	1
CONFIG_FS_CMD_FS_FORMAT	1
CONFIG_FS_CMD_FS_LIST	1
CONFIG_FS_CMD_FS_PARAMETERS_LIST	1
CONFIG_FS_CMD_FS_READ	1
CONFIG_FS_CMD_FS_REMOVE	1
CONFIG_FS_CMD_FS_WRITE	1
CONFIG_FS_CMD_I2C_READ	1
CONFIG_FS_CMD_I2C_WRITE	1
CONFIG_FS_CMD_LOG_LIST	1
CONFIG_FS_CMD_LOG_PRINT	1
CONFIG_FS_CMD_LOG_SET_LOG_MASK	1
CONFIG_FS_CMD_NETWORK_INTERFACE_LIST	1
CONFIG_FS_CMD_PING_PING	1
CONFIG_FS_CMD_PIN_READ	1
CONFIG_FS_CMD_PIN_SET_MODE	1
CONFIG_FS_CMD_PIN_WRITE	1
CONFIG_FS_CMD_SERVICE_LIST	1
CONFIG_FS_CMD_SERVICE_START	1
CONFIG_FS_CMD_SERVICE_STOP	1
CONFIG_FS_CMD_SETTINGS_LIST	1
CONFIG_FS_CMD_SETTINGS_READ	1
CONFIG_FS_CMD_SETTINGS_RESET	1
CONFIG_FS_CMD_SETTINGS_WRITE	1
CONFIG_FS_CMD_SYS_CONFIG	1
CONFIG_FS_CMD_SYS_INFO	1
CONFIG_FS_CMD_SYS_REBOOT	1
CONFIG_FS_CMD_SYS_UPTIME	1
CONFIG_FS_CMD_THRD_LIST	1
CONFIG_FS_CMD_THRD_SET_LOG_MASK	1
CONFIG_FS_CMD_USB_DEVICE_LIST	1
CONFIG_FS_CMD_USB_HOST_LIST	1
CONFIG_FS_PATH_MAX	64
CONFIG_MONITOR_THREAD	1
CONFIG_PREEMPTIVE_SCHEDULER	0
CONFIG_PROFILE_STACK	1
CONFIG_SETTINGS_AREA_SIZE	256
CONFIG_SHELL_COMMAND_MAX	64
CONFIG_SHELL_HISTORY_SIZE	768
CONFIG_SHELL_MINIMAL	0
CONFIG_SHELL_PROMPT	“\$ “
CONFIG_SOCKET_RAW	1
CONFIG_SPIFFS	1

Continued on next page

Table 1.9 – continued from previous page

Name	Value
CONFIG_START_CONSOLE	CONFIG_START_CONSOLE_UART
CONFIG_START_CONSOLE_DEVICE_INDEX	0
CONFIG_START_CONSOLE_UART_BAUDRATE	38400
CONFIG_START_CONSOLE_USB_CDC_CONTROL_INTERFACE	0
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_IN	2
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_OUT	3
CONFIG_START_CONSOLE_USB_CDC_WAIT_FOR_CONNETION	1
CONFIG_START_FILESYSTEM	1
CONFIG_START_FILESYSTEM_ADDRESS	0
CONFIG_START_FILESYSTEM_SIZE	65536
CONFIG_START_NETWORK	0
CONFIG_START_NETWORK_INTERFACE_WIFI_CONNECT_TIMEOUT	30
CONFIG_START_NETWORK_INTERFACE_WIFI_PASSWORD	MyWiFiPassword
CONFIG_START_NETWORK_INTERFACE_WIFI_SSID	MyWiFiSSID
CONFIG_START_SHELL	1
CONFIG_START_SHELL_PRIO	30
CONFIG_START_SHELL_STACK_SIZE	768
CONFIG_STD_OUTPUT_BUFFER_MAX	16
CONFIG_SYSTEM_TICK_FREQUENCY	100
CONFIG_SYS_CONFIG_STRING	1
CONFIG_SYS_SIMBA_MAIN_STACK_MAX	4096
CONFIG_THRD_CPU_USAGE	1
CONFIG_THRD_ENV	1
CONFIG_THRD_IDLE_STACK_SIZE	1024
CONFIG_THRD_SCHEDULED	1
CONFIG_THRD_TERMINATE	1
CONFIG_USB_DEVICE_PID	0x8037
CONFIG_USB_DEVICE_VID	0x2341

Homepage

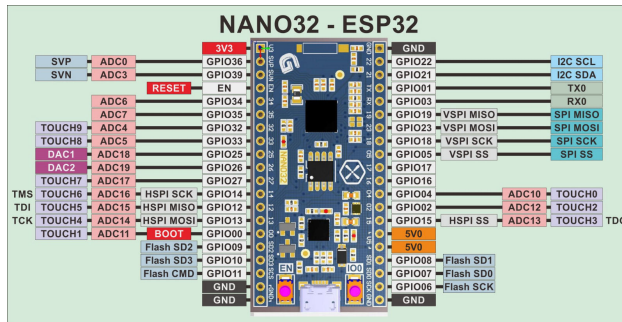
<http://www.kernel.org>

Mcu

linux

1.4.10 Nano32

Pinout



Default system features

The default configuration includes those major features. They are all initialized by `sys_start()` at the startup of the application.

- *Console.*
- File system.
- Networking.
- *Debug shell.*

Drivers

Supported drivers for this board.

- *can* — CAN bus
- *ds18b20* — One-wire temperature sensor
- *esp_wifi* — Espressif WiFi
- *flash* — Flash memory
- *owi* — One-Wire Interface
- *pin* — Digital pins
- *uart* — Universal Asynchronous Receiver/Transmitter

Library Reference

Read more about board specific functionality in the [Nano32](#) module documentation in the Library Reference.

Memory usage

Below is the memory usage of two applications:

- The [minimal-configuration](#) application is configured to only include the bare minimum of functionality for the low level kernel to run. That is, the thread scheduler and system tick.

- The `default-configuration` application is built with the default configuration, including a lot more functionality. See the list of *Default system features* above for a summary.

Application	Flash	RAM
minimal-configuration	179049	37876
default-configuration	307439	115100

Default configuration

Default Standard Library configuration.

Name	Value
CONFIG_ASSERT	1
CONFIG_DEBUG	1
CONFIG_FS_CMD_DS18B20_LIST	1
CONFIG_FS_CMD_ESP_WIFI_STATUS	0
CONFIG_FS_CMD_FS_APPEND	1
CONFIG_FS_CMD_FS_COUNTERS_LIST	1
CONFIG_FS_CMD_FS_COUNTERS_RESET	1
CONFIG_FS_CMD_FS_FILESYSTEMS_LIST	1
CONFIG_FS_CMD_FS_FORMAT	1
CONFIG_FS_CMD_FS_LIST	1
CONFIG_FS_CMD_FS_PARAMETERS_LIST	1
CONFIG_FS_CMD_FS_READ	1
CONFIG_FS_CMD_FS_REMOVE	1
CONFIG_FS_CMD_FS_WRITE	1
CONFIG_FS_CMD_I2C_READ	1
CONFIG_FS_CMD_I2C_WRITE	1
CONFIG_FS_CMD_LOG_LIST	1
CONFIG_FS_CMD_LOG_PRINT	1
CONFIG_FS_CMD_LOG_SET_LOG_MASK	1
CONFIG_FS_CMD_NETWORK_INTERFACE_LIST	1
CONFIG_FS_CMD_PING_PING	1
CONFIG_FS_CMD_PIN_READ	1
CONFIG_FS_CMD_PIN_SET_MODE	1
CONFIG_FS_CMD_PIN_WRITE	1
CONFIG_FS_CMD_SERVICE_LIST	1
CONFIG_FS_CMD_SERVICE_START	1
CONFIG_FS_CMD_SERVICE_STOP	1
CONFIG_FS_CMD_SETTINGS_LIST	1
CONFIG_FS_CMD_SETTINGS_READ	1
CONFIG_FS_CMD_SETTINGS_RESET	1
CONFIG_FS_CMD_SETTINGS_WRITE	1
CONFIG_FS_CMD_SYS_CONFIG	1
CONFIG_FS_CMD_SYS_INFO	1
CONFIG_FS_CMD_SYS_REBOOT	1
CONFIG_FS_CMD_SYS_UPTIME	1
CONFIG_FS_CMD_THRD_LIST	1
CONFIG_FS_CMD_THRD_SET_LOG_MASK	1
CONFIG_FS_CMD_USB_DEVICE_LIST	1
CONFIG_FS_CMD_USB_HOST_LIST	1

Continued on next page

Table 1.10 – continued from previous page

Name	Value
CONFIG_FS_PATH_MAX	64
CONFIG_MONITOR_THREAD	0
CONFIG_PREEMPTIVE_SCHEDULER	0
CONFIG_PROFILE_STACK	1
CONFIG_SETTINGS_AREA_SIZE	256
CONFIG_SHELL_COMMAND_MAX	64
CONFIG_SHELL_HISTORY_SIZE	768
CONFIG_SHELL_MINIMAL	0
CONFIG_SHELL_PROMPT	“\$ “
CONFIG_SOCKET_RAW	1
CONFIG_SPIFFS	1
CONFIG_START_CONSOLE	CONFIG_START_CONSOLE_UART
CONFIG_START_CONSOLE_DEVICE_INDEX	0
CONFIG_START_CONSOLE_UART_BAUDRATE	115200
CONFIG_START_CONSOLE_USB_CDC_CONTROL_INTERFACE	0
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_IN	2
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_OUT	3
CONFIG_START_CONSOLE_USB_CDC_WAIT_FOR_CONNETION	1
CONFIG_START_FILESYSTEM	1
CONFIG_START_FILESYSTEM_ADDRESS	0x00300000
CONFIG_START_FILESYSTEM_SIZE	32768
CONFIG_START_NETWORK	1
CONFIG_START_NETWORK_INTERFACE_WIFI_CONNECT_TIMEOUT	30
CONFIG_START_NETWORK_INTERFACE_WIFI_PASSWORD	MyWiFiPassword
CONFIG_START_NETWORK_INTERFACE_WIFI_SSID	MyWiFiSSID
CONFIG_START_SHELL	1
CONFIG_START_SHELL_PRIO	30
CONFIG_START_SHELL_STACK_SIZE	4096
CONFIG_STD_OUTPUT_BUFFER_MAX	16
CONFIG_SYSTEM_TICK_FREQUENCY	100
CONFIG_SYS_CONFIG_STRING	1
CONFIG_SYS_SIMBA_MAIN_STACK_MAX	4096
CONFIG_THRD_CPU_USAGE	1
CONFIG_THRD_ENV	1
CONFIG_THRD_IDLE_STACK_SIZE	1024
CONFIG_THRD_SCHEDULED	1
CONFIG_THRD_TERMINATE	1
CONFIG_USB_DEVICE_PID	0x8037
CONFIG_USB_DEVICE VID	0x2341

Homepage

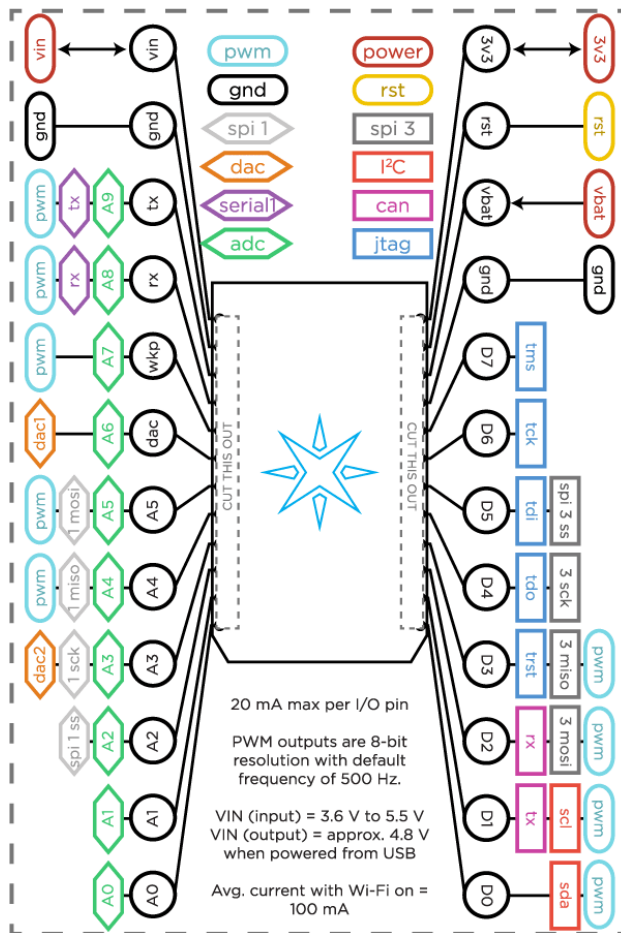
<http://esp32.de>

Mcu

esp32

1.4.11 Particle IO Photon

Pinout



Default system features

The default configuration includes those major features. They are all initialized by `sys_start()` at the startup of the application.

- *Console.*
- *Debug shell.*

Drivers

Supported drivers for this board.

- *bcm43362* — *BCM43362*
- *flash* — *Flash memory*
- *i2c_soft* — *Software I2C*
- *pin* — *Digital pins*

- *sdio* — *Secure Digital Input Output*
- *uart* — *Universal Asynchronous Receiver/Transmitter*

Library Reference

Read more about board specific functionality in the *Particle IO Photon* module documentation in the Library Reference.

Memory usage

Below is the memory usage of two applications:

- The *minimal-configuration* application is configured to only include the bare minimum of functionality for the low level kernel to run. That is, the thread scheduler and system tick.
- The *default-configuration* application is built with the default configuration, including a lot more functionality. See the list of *Default system features* above for a summary.

Application	Flash	RAM
minimal-configuration	16604	3376
default-configuration	60268	6746

Default configuration

Default Standard Library configuration.

Name	Value
CONFIG_ASSERT	1
CONFIG_DEBUG	1
CONFIG_FS_CMD_DS18B20_LIST	1
CONFIG_FS_CMD_ESP_WIFI_STATUS	0
CONFIG_FS_CMD_FS_APPEND	1
CONFIG_FS_CMD_FS_COUNTERS_LIST	1
CONFIG_FS_CMD_FS_COUNTERS_RESET	1
CONFIG_FS_CMD_FS_FILESYSTEMS_LIST	1
CONFIG_FS_CMD_FS_FORMAT	1
CONFIG_FS_CMD_FS_LIST	1
CONFIG_FS_CMD_FS_PARAMETERS_LIST	1
CONFIG_FS_CMD_FS_READ	1
CONFIG_FS_CMD_FS_REMOVE	1
CONFIG_FS_CMD_FS_WRITE	1
CONFIG_FS_CMD_I2C_READ	1
CONFIG_FS_CMD_I2C_WRITE	1
CONFIG_FS_CMD_LOG_LIST	1
CONFIG_FS_CMD_LOG_PRINT	1
CONFIG_FS_CMD_LOG_SET_LOG_MASK	1
CONFIG_FS_CMD_NETWORK_INTERFACE_LIST	1
CONFIG_FS_CMD_PING_PING	1
CONFIG_FS_CMD_PIN_READ	1
CONFIG_FS_CMD_PIN_SET_MODE	1
CONFIG_FS_CMD_PIN_WRITE	1

Continued on next page

Table 1.11 – continued from previous page

Name	Value
CONFIG_FS_CMD_SERVICE_LIST	1
CONFIG_FS_CMD_SERVICE_START	1
CONFIG_FS_CMD_SERVICE_STOP	1
CONFIG_FS_CMD_SETTINGS_LIST	1
CONFIG_FS_CMD_SETTINGS_READ	1
CONFIG_FS_CMD_SETTINGS_RESET	1
CONFIG_FS_CMD_SETTINGS_WRITE	1
CONFIG_FS_CMD_SYS_CONFIG	1
CONFIG_FS_CMD_SYS_INFO	1
CONFIG_FS_CMD_SYS_REBOOT	1
CONFIG_FS_CMD_SYS_UPTIME	1
CONFIG_FS_CMD_THRD_LIST	1
CONFIG_FS_CMD_THRD_SET_LOG_MASK	1
CONFIG_FS_CMD_USB_DEVICE_LIST	1
CONFIG_FS_CMD_USB_HOST_LIST	1
CONFIG_FS_PATH_MAX	64
CONFIG_MONITOR_THREAD	1
CONFIG_PREEMPTIVE_SCHEDULER	0
CONFIG_PROFILE_STACK	1
CONFIG_SETTINGS_AREA_SIZE	256
CONFIG_SHELL_COMMAND_MAX	64
CONFIG_SHELL_HISTORY_SIZE	768
CONFIG_SHELL_MINIMAL	0
CONFIG_SHELL_PROMPT	“\$ “
CONFIG_SOCKET_RAW	1
CONFIG_SPIFFS	0
CONFIG_START_CONSOLE	CONFIG_START_CONSOLE_UART
CONFIG_START_CONSOLE_DEVICE_INDEX	0
CONFIG_START_CONSOLE_UART_BAUDRATE	38400
CONFIG_START_CONSOLE_USB_CDC_CONTROL_INTERFACE	0
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_IN	2
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_OUT	3
CONFIG_START_CONSOLE_USB_CDC_WAIT_FOR_CONNENTION	1
CONFIG_START_FILESYSTEM	0
CONFIG_START_FILESYSTEM_ADDRESS	0
CONFIG_START_FILESYSTEM_SIZE	65536
CONFIG_START_NETWORK	0
CONFIG_START_NETWORK_INTERFACE_WIFI_CONNECT_TIMEOUT	30
CONFIG_START_NETWORK_INTERFACE_WIFI_PASSWORD	MyWiFiPassword
CONFIG_START_NETWORK_INTERFACE_WIFI_SSID	MyWiFiSSID
CONFIG_START_SHELL	1
CONFIG_START_SHELL_PRIO	30
CONFIG_START_SHELL_STACK_SIZE	768
CONFIG_STD_OUTPUT_BUFFER_MAX	16
CONFIG_SYSTEM_TICK_FREQUENCY	100
CONFIG_SYS_CONFIG_STRING	1
CONFIG_SYS_SIMBA_MAIN_STACK_MAX	4096
CONFIG_THRD_CPU_USAGE	1
CONFIG_THRD_ENV	1

Continued on next page

Table 1.11 – continued from previous page

Name	Value
CONFIG_THRD_IDLE_STACK_SIZE	384
CONFIG_THRD_SCHEDULED	1
CONFIG_THRD_TERMINATE	1
CONFIG_USB_DEVICE_PID	0x8037
CONFIG_USB_DEVICE_VID	0x2341

Homepage

<https://docs.particle.io/datasheets/photon-datasheet/>

Mcu

stm32f205rg

Detailed pinout

Right side pins

USB	Pin	Exposed Functions				STM32 Pin	PØ Pin #	PØ Pin Name
P H O T O N	3V3	3V3						
	RST	RST				E8	26	MICRO_RST_N
	VBAT	VBAT				A9	28	VBAT
	GND	GND						
	D7	JTAG_TMS				PA13	44	MICRO_JTAG_TMS
	D6	JTAG_TCK				PA14	40	MICRO_JTAG_TCK
	D5	JTAG_TDI	SPI3_SS		I2S3_WS	PA15	43	MICRO_JTAG_TDI
	D4	JTAG_TDO	SPI3_SCK		I2S3_SCK	PB3	41	MICRO_JTAG_TDO
	D3	JTAG_TRST	SPI3_MISO	TIM3_CH1		PB4	42	MICRO_JTAG_TRSTN
	D2		SPI3_MOSI	CAN2_RX	TIM3_CH2	PB5	3	MICRO_GPIO_5
	D1	SCL		CAN2_TX	TIM4_CH1	PB6	5	MICRO_GPIO_3
	D0	SDA			TIM4_CH2	PB7	4	MICRO_GPIO_4

Left side pins

Pin	USB	Exposed Functions				STM32 Pin	PØ Pin #	PØ Pin Name
VIN	P H O T O N	VIN						
GND		GND						
TX			USART1_TX	TIM1_CH2		PA9	39	MICRO_UART_TX
RX			USART1_RX	TIM1_CH3		PA10	38	MICRO_UART_RX
WKP		ADC0		TIM5_CH1		PA0	27	MICRO_WKUP
DAC		ADC4			DAC1	PA4	22	MICRO_SPI_SSN
A5		ADC7	SPI1_MOSI	TIM3_CH2		PA7	23	MICRO_SPI_MOSI
A4		ADC6	SPI1_MISO	TIM3_CH1		PA6	25	MICRO_SPI_MISO
A3		ADC5	SPI1_SCK		DAC2	PA5	24	MICRO_SPI_SCK
A2		ADC12	SPI1_SS			PC2	2	MICRO_GPIO_6
A1		ADC13				PC3	1	MICRO_GPIO_7
A0		ADC15				PC5	54	MICRO_GPIO_8

User I/O

	User I/O	Photon Pin #	Exposed Functions			STM32 Pin	PØ Pin #	PØ Pin Name
P H O T O N	RGB LED - RED	27		TIM2_CH2		PA1	8	MICRO_GPIO_0
	RGB LED - GREEN	28		TIM2_CH3		PA2	7	MICRO_GPIO_1
	RGB LED - BLUE	29		TIM2_CH4		PA3	6	MICRO_GPIO_2
	Setup Button	26		TIM3_CH2	I2S3_MCK	PC7	53	MICRO_GPIO_9
	Reset Button	23				E8	26	MICRO_RST_N
	USB Data+	31				PB15	51	MICRO_USB_HS_DP
	USB Data-	30				PB14	52	MICRO_USB_HS_DM
	SMPS Enable	25						
	Peripheral Key	ADC	SPI	PWM/Servo/Tone				
		JTAG	SPI1	I2S	DAC			
		I2C/Wire	Serial1	CAN				

Prerequisites

Install the dfu-utility.

```
git clone git://git.code.sf.net/p/dfu-util/dfu-util
cd dfu-util
sudo apt-get build-dep dfu-util
./autogen.sh
./configure
make
sudo make install
cd ..

# Give users access to the device.
sudo cp simba/environment/udev/49-photon.rules /etc/udev/rules.d
```

Flashing

The Photon must enter DFU mode before software can be uploaded to it. It's recommended to use the manual method to verify that software can be successfully uploaded to the board, and then start using the automatic method to reduce the manual work for each software upload.

Automatic (recommended)

- Connect DTR on the serial adapter to the RST pin on the Photon.
- Connect RTS on the serial adapter to the SETUP pad on the bottom side of the Photon. This requires soldering a cable to the SETUP pad.

Upload the software with `make BOARD=photon upload`.

Manual

To enter DFU Mode:

1. Hold down the RESET and SETUP buttons.
2. Release only the RESET button, while holding down the SETUP button.
3. Wait for the LED to start flashing yellow (it will flash magenta first).
4. Release the SETUP button.

NOTE: Do **not** connect DTR and/or RTS using manual upload. They must only be connected using the automatic method.

Upload the software with `make BOARD=photon upload`.

Library Reference

Read more about board specific functionality in the *STM32F3DISCOVERY* module documentation in the Library Reference.

Memory usage

Below is the memory usage of two applications:

- The [minimal-configuration](#) application is configured to only include the bare minimum of functionality for the low level kernel to run. That is, the thread scheduler and system tick.
- The [default-configuration](#) application is built with the default configuration, including a lot more functionality. See the list of *Default system features* above for a summary.

Application	Flash	RAM
minimal-configuration	16344	2896
default-configuration	58988	6254

Default configuration

Default Standard Library configuration.

Name	Value
CONFIG_ASSERT	1
CONFIG_DEBUG	1
CONFIG_FS_CMD_DS18B20_LIST	1
CONFIG_FS_CMD_ESP_WIFI_STATUS	0
CONFIG_FS_CMD_FS_APPEND	1
CONFIG_FS_CMD_FS_COUNTERS_LIST	1
CONFIG_FS_CMD_FS_COUNTERS_RESET	1
CONFIG_FS_CMD_FS_FILESYSTEMS_LIST	1
CONFIG_FS_CMD_FS_FORMAT	1
CONFIG_FS_CMD_FS_LIST	1
CONFIG_FS_CMD_FS_PARAMETERS_LIST	1
CONFIG_FS_CMD_FS_READ	1
CONFIG_FS_CMD_FS_REMOVE	1
CONFIG_FS_CMD_FS_WRITE	1
CONFIG_FS_CMD_I2C_READ	1
CONFIG_FS_CMD_I2C_WRITE	1
CONFIG_FS_CMD_LOG_LIST	1
CONFIG_FS_CMD_LOG_PRINT	1
CONFIG_FS_CMD_LOG_SET_LOG_MASK	1
CONFIG_FS_CMD_NETWORK_INTERFACE_LIST	1
CONFIG_FS_CMD_PING_PING	1
CONFIG_FS_CMD_PIN_READ	1
CONFIG_FS_CMD_PIN_SET_MODE	1
CONFIG_FS_CMD_PIN_WRITE	1
CONFIG_FS_CMD_SERVICE_LIST	1
CONFIG_FS_CMD_SERVICE_START	1
CONFIG_FS_CMD_SERVICE_STOP	1
CONFIG_FS_CMD_SETTINGS_LIST	1

Continued on next page

Table 1.12 – continued from previous page

Name	Value
CONFIG_FS_CMD_SETTINGS_READ	1
CONFIG_FS_CMD_SETTINGS_RESET	1
CONFIG_FS_CMD_SETTINGS_WRITE	1
CONFIG_FS_CMD_SYS_CONFIG	1
CONFIG_FS_CMD_SYS_INFO	1
CONFIG_FS_CMD_SYS_REBOOT	1
CONFIG_FS_CMD_SYS_UPTIME	1
CONFIG_FS_CMD_THRD_LIST	1
CONFIG_FS_CMD_THRD_SET_LOG_MASK	1
CONFIG_FS_CMD_USB_DEVICE_LIST	1
CONFIG_FS_CMD_USB_HOST_LIST	1
CONFIG_FS_PATH_MAX	64
CONFIG_MONITOR_THREAD	1
CONFIG_PREEMPTIVE_SCHEDULER	0
CONFIG_PROFILE_STACK	1
CONFIG_SETTINGS_AREA_SIZE	256
CONFIG_SHELL_COMMAND_MAX	64
CONFIG_SHELL_HISTORY_SIZE	768
CONFIG_SHELL_MINIMAL	0
CONFIG_SHELL_PROMPT	“\$ “
CONFIG_SOCKET_RAW	1
CONFIG_SPIFFS	0
CONFIG_START_CONSOLE	CONFIG_START_CONSOLE_UART
CONFIG_START_CONSOLE_DEVICE_INDEX	0
CONFIG_START_CONSOLE_UART_BAUDRATE	38400
CONFIG_START_CONSOLE_USB_CDC_CONTROL_INTERFACE	0
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_IN	2
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_OUT	3
CONFIG_START_CONSOLE_USB_CDC_WAIT_FOR_CONNENTION	1
CONFIG_START_FILESYSTEM	0
CONFIG_START_FILESYSTEM_ADDRESS	0
CONFIG_START_FILESYSTEM_SIZE	65536
CONFIG_START_NETWORK	0
CONFIG_START_NETWORK_INTERFACE_WIFI_CONNECT_TIMEOUT	30
CONFIG_START_NETWORK_INTERFACE_WIFI_PASSWORD	MyWiFiPassword
CONFIG_START_NETWORK_INTERFACE_WIFI_SSID	MyWiFiSSID
CONFIG_START_SHELL	1
CONFIG_START_SHELL_PRIO	30
CONFIG_START_SHELL_STACK_SIZE	768
CONFIG_STD_OUTPUT_BUFFER_MAX	16
CONFIG_SYSTEM_TICK_FREQUENCY	100
CONFIG_SYS_CONFIG_STRING	1
CONFIG_SYS_SIMBA_MAIN_STACK_MAX	4096
CONFIG_THRD_CPU_USAGE	1
CONFIG_THRD_ENV	1
CONFIG_THRD_IDLE_STACK_SIZE	384
CONFIG_THRD_SCHEDULED	1
CONFIG_THRD_TERMINATE	1
CONFIG_USB_DEVICE_PID	0x8037

Continued on next page

Table 1.12 – continued from previous page

Name	Value
CONFIG_USB_DEVICE_VID	0x2341

Homepage

http://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-eval-tools/stm32-mcu-eval-tools/stm32-mcu-discovery-kits/stm32f3discovery.html

Mcu

stm32f303vc

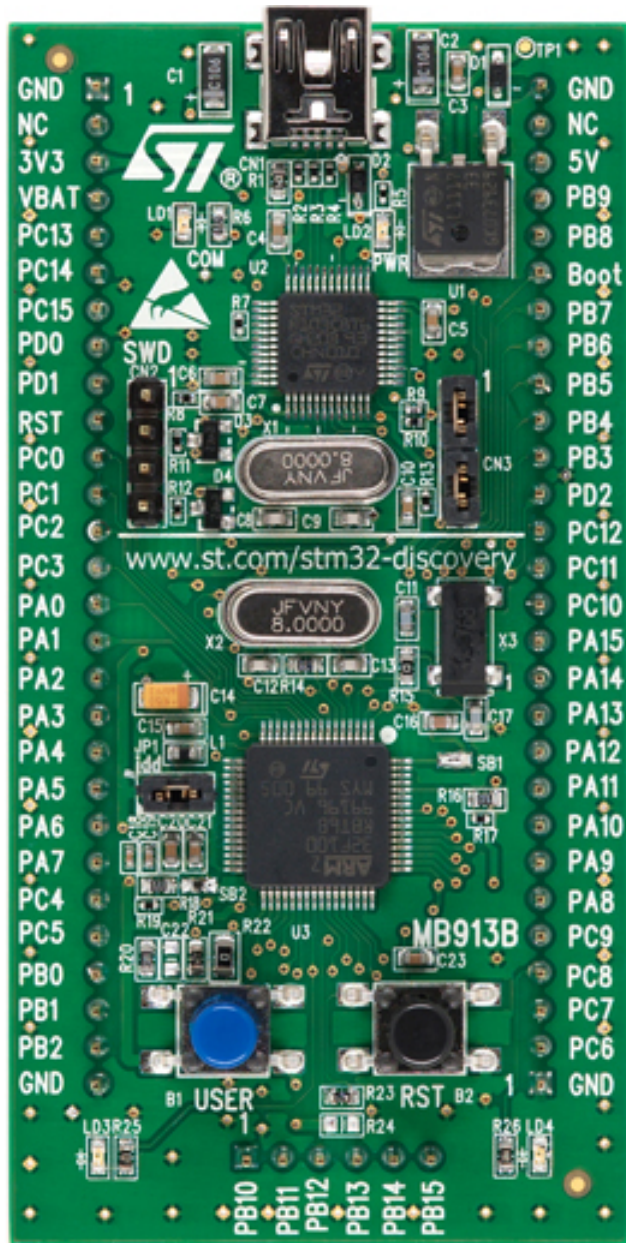
Pin functions

These are the default pin functions in Simba.

Function	Pin
UART0 TX	PA9
UART0 RX	PA10
UART1 TX	PA2
UART1 RX	PA3
UART2 TX	PB10
UART2 RX	PB11
SPI0 SCK	PA5
SPI0 MISO	PA6
SPI0 MOSI	PA7
SPI1 SCK	PA13
SPI1 MISO	PA14
SPI1 MOSI	PA15
SPI2 SCK	PC10
SPI2 MISO	PC11
SPI2 MOSI	PC12
I2C0 SCL	PB8
I2C0 SDA	PB9
I2C1 SCL	PF0
I2C1 SDA	PF1
CAN TX	PD1
CAN RX	PD0

1.4.13 STM32VLDISCOVERY

Pinout



Default system features

The default configuration includes those major features. They are all initialized by `sys_start()` at the startup of the application.

- *Console.*
- *Debug shell.*

Drivers

Supported drivers for this board.

- *flash* — *Flash memory*
- *i2c_soft* — *Software I2C*
- *pin* — *Digital pins*
- *uart* — *Universal Asynchronous Receiver/Transmitter*

Library Reference

Read more about board specific functionality in the *STM32VLDISCOVERY* module documentation in the Library Reference.

Memory usage

Below is the memory usage of two applications:

- The *minimal-configuration* application is configured to only include the bare minimum of functionality for the low level kernel to run. That is, the thread scheduler and system tick.
- The *default-configuration* application is built with the default configuration, including a lot more functionality. See the list of *Default system features* above for a summary.

Application	Flash	RAM
minimal-configuration	17036	3052
default-configuration	59808	6290

Default configuration

Default Standard Library configuration.

Name	Value
CONFIG_ASSERT	1
CONFIG_DEBUG	1
CONFIG_FS_CMD_DS18B20_LIST	1
CONFIG_FS_CMD_ESP_WIFI_STATUS	0
CONFIG_FS_CMD_FS_APPEND	1
CONFIG_FS_CMD_FS_COUNTERS_LIST	1
CONFIG_FS_CMD_FS_COUNTERS_RESET	1
CONFIG_FS_CMD_FS_FILESYSTEMS_LIST	1
CONFIG_FS_CMD_FS_FORMAT	1
CONFIG_FS_CMD_FS_LIST	1
CONFIG_FS_CMD_FS_PARAMETERS_LIST	1
CONFIG_FS_CMD_FS_READ	1
CONFIG_FS_CMD_FS_REMOVE	1
CONFIG_FS_CMD_FS_WRITE	1
CONFIG_FS_CMD_I2C_READ	1
CONFIG_FS_CMD_I2C_WRITE	1
CONFIG_FS_CMD_LOG_LIST	1
Continued on next page	

Table 1.13 – continued from previous page

Name	Value
CONFIG_FS_CMD_LOG_PRINT	1
CONFIG_FS_CMD_LOG_SET_LOG_MASK	1
CONFIG_FS_CMD_NETWORK_INTERFACE_LIST	1
CONFIG_FS_CMD_PING_PING	1
CONFIG_FS_CMD_PIN_READ	1
CONFIG_FS_CMD_PIN_SET_MODE	1
CONFIG_FS_CMD_PIN_WRITE	1
CONFIG_FS_CMD_SERVICE_LIST	1
CONFIG_FS_CMD_SERVICE_START	1
CONFIG_FS_CMD_SERVICE_STOP	1
CONFIG_FS_CMD_SETTINGS_LIST	1
CONFIG_FS_CMD_SETTINGS_READ	1
CONFIG_FS_CMD_SETTINGS_RESET	1
CONFIG_FS_CMD_SETTINGS_WRITE	1
CONFIG_FS_CMD_SYS_CONFIG	1
CONFIG_FS_CMD_SYS_INFO	1
CONFIG_FS_CMD_SYS_REBOOT	1
CONFIG_FS_CMD_SYS_UPTIME	1
CONFIG_FS_CMD_THRD_LIST	1
CONFIG_FS_CMD_THRD_SET_LOG_MASK	1
CONFIG_FS_CMD_USB_DEVICE_LIST	1
CONFIG_FS_CMD_USB_HOST_LIST	1
CONFIG_FS_PATH_MAX	64
CONFIG_MONITOR_THREAD	1
CONFIG_PREEMPTIVE_SCHEDULER	0
CONFIG_PROFILE_STACK	1
CONFIG_SETTINGS_AREA_SIZE	256
CONFIG_SHELL_COMMAND_MAX	64
CONFIG_SHELL_HISTORY_SIZE	768
CONFIG_SHELL_MINIMAL	0
CONFIG_SHELL_PROMPT	“\$ “
CONFIG_SOCKET_RAW	1
CONFIG_SPIFFS	0
CONFIG_START_CONSOLE	CONFIG_START_CONSOLE_UART
CONFIG_START_CONSOLE_DEVICE_INDEX	0
CONFIG_START_CONSOLE_UART_BAUDRATE	38400
CONFIG_START_CONSOLE_USB_CDC_CONTROL_INTERFACE	0
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_IN	2
CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_OUT	3
CONFIG_START_CONSOLE_USB_CDC_WAIT_FOR_CONNCTION	1
CONFIG_START_FILESYSTEM	0
CONFIG_START_FILESYSTEM_ADDRESS	0
CONFIG_START_FILESYSTEM_SIZE	65536
CONFIG_START_NETWORK	0
CONFIG_START_NETWORK_INTERFACE_WIFI_CONNECT_TIMEOUT	30
CONFIG_START_NETWORK_INTERFACE_WIFI_PASSWORD	MyWiFiPassword
CONFIG_START_NETWORK_INTERFACE_WIFI_SSID	MyWiFiSSID
CONFIG_START_SHELL	1
CONFIG_START_SHELL_PRIO	30

Continued on next page

Table 1.13 – continued from previous page

Name	Value
CONFIG_START_SHELL_STACK_SIZE	768
CONFIG_STD_OUTPUT_BUFFER_MAX	16
CONFIG_SYSTEM_TICK_FREQUENCY	100
CONFIG_SYS_CONFIG_STRING	1
CONFIG_SYS_SIMBA_MAIN_STACK_MAX	4096
CONFIG_THRD_CPU_USAGE	1
CONFIG_THRD_ENV	1
CONFIG_THRD_IDLE_STACK_SIZE	384
CONFIG_THRD_SCHEDULED	1
CONFIG_THRD_TERMINATE	1
CONFIG_USB_DEVICE_PID	0x8037
CONFIG_USB_DEVICE_VID	0x2341

Homepage

http://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-eval-tools/stm32-mcu-eval-tools/stm32-mcu-discovery-kits/stm32vldiscovery.html?sc=internet/evalboard/product/250863.jsp

Mcu

stm32f100rb

st-link

```
sudo apt install libusb-1.0-0-dev
git clone https://github.com/eerimoq/stlink
./autogen.sh
./configure
make
sudo cp etc/udev/rules.d/49* /etc/udev/rules.d
udevadm control --reload-rules
udevadm trigger

modprobe -r usb-storage && modprobe usb-storage quirks=483:3744:i

st-util -l
arm-none-eabi-gdb app.out
$ target extended-remote localhost:4242
```

Plug in the board in the PC.

Pin functions

These are the default pin functions in Simba.

Function	Pin
UART0 TX	PA9
UART0 RX	PA10
UART1 TX	PA2
UART1 RX	PA3
UART2 TX	PC10
UART2 RX	PC11
SPI0 SCK	PA5
SPI0 MISO	PA6
SPI0 MOSI	PA7
I2C0 SCL	PB8
I2C0 SDA	PB9

1.5 Examples

Below is a list of simple examples that are useful to understand the basics of *Simba*.

There are a lot more [examples](#) and [unit tests](#) on Github that shows how to use most of the *Simba* modules.

1.5.1 Analog Read

About

Read the value of an analog pin periodically once every second and print the read value to standard output.

Source code

```
/**
 * @section License
 *
 * The MIT License (MIT)
 *
 * Copyright (c) 2014-2016, Erik Moqvist
 *
 * 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.
*
* This file is part of the Simba project.
*/

#include "simba.h"

int main()
{
    int value;
    struct analog_input_pin_t pin;

    sys_start();
    analog_input_pin_module_init();

    /* Initialize the analog input pin. */
    analog_input_pin_init(&pin, &pin_a0_dev);

    while (1) {
        /* Read the analog pin value and print it. */
        value = analog_input_pin_read(&pin);
        std_printf(FSTR("value = %d\r\n"), value);

        /* Wait 100 ms. */
        thrd_sleep_ms(100);
    }

    return (0);
}
```

The source code can also be found on Github in the [examples/analog_read](#) folder.

Build and run

Build and run the application.

```
$ cd examples/analog_read
$ make -s BOARD=<board> run
value = 234
value = 249
value = 230
```

1.5.2 Analog Write

About

Write analog values to an analog output pin to form a sawtooth wave. Connect a LED to the analog output pin and watch the brightness of the LED change.

Source code

```
/**
 * @section License
```



```

*
* The MIT License (MIT)
*
* Copyright (c) 2014-2016, Erik Moqvist
*
* 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.
*
* This file is part of the Simba project.
*/

#include "simba.h"

int main()
{
    int value;
    struct analog_output_pin_t pin;

    sys_start();
    analog_output_pin_module_init();

    /* Initialize the analog output pin. */
    analog_output_pin_init(&pin, &pin_d10_dev);

    value = 0;

    while (1) {
        /* Write a sawtooth wave to the analog output pin. */
        analog_output_pin_write(&pin, value);
        value += 5;
        value %= 1024;

        /* Wait ten milliseconds. */
        thrd_sleep_ms(10);
    }

    return (0);
}

```

The source code can also be found on Github in the [examples/analog_write](#) folder.

Build and run

Build and upload the application.

```
$ cd examples/analog_write
$ make -s BOARD=<board> upload
```

1.5.3 Blink

About

Turn a LED on and off periodically once a second. This example illustrates how to use digital pins and sleep a thread.

Source code

```
/**
 * @section License
 *
 * The MIT License (MIT)
 *
 * Copyright (c) 2014-2016, Erik Moqvist
 *
 * 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.
 *
 * This file is part of the Simba project.
 */

#include "simba.h"

int main()
{
    struct pin_driver_t led;

    /* Start the system. */
    sys_start();

    /* Initialize the LED pin as output and set its value to 1. */
```

```

pin_init(&led, &pin_led_dev, PIN_OUTPUT);
pin_write(&led, 1);

while (1) {
    /* Wait half a second. */
    thrd_sleep_ms(500);

    /* Toggle the LED on/off. */
    pin_toggle(&led);
}

return (0);
}

```

The source code can also be found on Github in the [examples/blink](#) folder.

Build and run

Build and upload the application.

```

$ cd examples/blink
$ make -s BOARD=<board> upload

```

1.5.4 DS18B20

About

Read and print the temperature measured with one or more DS18B20 sensors.

Source code

```

/**
 * @section License
 *
 * The MIT License (MIT)
 *
 * Copyright (c) 2014-2016, Erik Moqvist
 *
 * 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.
*
* This file is part of the Simba project.
*/

#include "simba.h"

int main()
{
    struct owi_driver_t owi;
    struct ds18b20_driver_t ds;
    struct owi_device_t devices[4];
    char temperature[16], *temperature_p;
    int number_of_sensors;
    int i;

    /* Initialization. */
    sys_start();
    ds18b20_module_init();
    owi_init(&owi, &pin_d7_dev, devices, membersof(devices));
    ds18b20_init(&ds, &owi);
    time_busy_wait_us(50000);

    /* Search for devices on the OWI bus. */
    number_of_sensors = owi_search(&owi);
    std_printf(FSTR("Number of sensors: %d\r\n"), number_of_sensors);

    while (1) {
        /* Take a new temperature sample. */
        ds18b20_convert(&ds);

        for (i = 0; i < owi.len; i++) {
            if (devices[i].id[0] != DS18B20_FAMILY_CODE) {
                continue;
            }

            temperature_p = ds18b20_get_temperature_str(&ds,
                                                         devices[i].id,
                                                         temperature);

            std_printf(FSTR("Device id: %02x %02x %02x %02x %02x %02x %02x %02x,"
                           " Temperature: %s\r\n"),
                      (unsigned int)devices[i].id[0],
                      (unsigned int)devices[i].id[1],
                      (unsigned int)devices[i].id[2],
                      (unsigned int)devices[i].id[3],
                      (unsigned int)devices[i].id[4],
                      (unsigned int)devices[i].id[5],
                      (unsigned int)devices[i].id[6],
                      (unsigned int)devices[i].id[7],
                      temperature_p);
        }
    }

    return (0);
}
```

```
}

```

The source code can also be found on Github in the [examples/ds18b20](#) folder.

Build and run

Build and run the application.

```
$ cd examples/ds18b20
$ make -s BOARD=<board> run
Number of sensors: 2
Device id: 28 9c 1d 5d 05 00 00 32, Temperature: 22.6250
Device id: 28 95 32 5d 05 00 00 33, Temperature: 22.6875
```

1.5.5 Filesystem

About

Create the file `counter.txt` and write 0 to it. Everytime the application is restarted the counter is incremented by one.

Source code

```
/**
 * @section License
 *
 * The MIT License (MIT)
 *
 * Copyright (c) 2014-2016, Erik Moqvist
 *
 * 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.
 *
 * This file is part of the Simba project.
 */
```

```
#include "simba.h"

#if !defined(BOARD_ARDUINO_DUE) && !defined(ARCH_ESP) && !defined(ARCH_ESP32)
#   error "This example can only be built for Arduino Due, ESP and ESP32."
#endif

/**
 * Increment the counter in 'counter.txt'.
 */
static int increment_counter(void)
{
    char buf[32];
    struct fs_file_t file;
    long counter;
    size_t size;

    std_printf(FSTR("Incrementing the counter in 'counter.txt'.\r\n"));

    if (fs_open(&file, "counter.txt", FS_RDWR) != 0) {
        /* Create the file if missing. */
        if (fs_open(&file,
                    "counter.txt",
                    FS_CREAT | FS_TRUNC | FS_RDWR) != 0) {
            return (-1);
        }

        if (fs_write(&file, "0", 2) != 2) {
            return (-2);
        }

        if (fs_seek(&file, 0, FS_SEEK_SET) != 0) {
            return (-3);
        }
    }

    if (fs_read(&file, buf, 16) <= 0) {
        return (-4);
    }

    if (std_strtol(buf, &counter) == NULL) {
        return (-5);
    }

    /* Increment the counter. */
    counter++;
    std_sprintf(buf, FSTR("%lu"), counter);
    size = strlen(buf) + 1;

    if (fs_seek(&file, 0, FS_SEEK_SET) != 0) {
        return (-6);
    }

    if (fs_write(&file, buf, size) != size) {
        return (-7);
    }

    if (fs_close(&file) != 0) {
        return (-8);
    }
}
```

```

    }

    std_printf(FSTR("Counter incremented to %lu\r\n"), counter);

    return (0);
}

int main()
{
    int res;

    sys_start();
    std_printf(sys_get_info());

    /* Increment the counter. */
    res = increment_counter();

    if (res != 0) {
        std_printf(FSTR("Failed to increment the counter with error %d.\r\n"),
                    res);
    }

    /* The shell thread is started in sys_start() so just suspend this
       thread. */
    thrd_suspend(NULL);

    return (0);
}

```

The source code can also be found on Github in the [examples/filesystem](#) folder.

Build and run

Build and run the application.

```

$ cd examples/filesystem
$ make -s BOARD=arduino_due upload

```

The output in the terminal emulator:

```

Incrementing the counter in 'counter.txt'.
Counter incremented to 1.
<manually reset the board>
Incrementing the counter in 'counter.txt'.
Counter incremented to 2.
<manually reset the board>
Incrementing the counter in 'counter.txt'.
Counter incremented to 3.

```

1.5.6 Hello World

About

This application prints “Hello world!” to standard output.

Source code

```
/**
 * @section License
 *
 * The MIT License (MIT)
 *
 * Copyright (c) 2014-2016, Erik Moqvist
 *
 * 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.
 *
 * This file is part of the Simba project.
 */

#include "simba.h"

int main()
{
    /* Start the system. */
    sys_start();

    std_printf(FSTR("Hello world!\r\n"));

    return (0);
}
```

The source code can also be found on Github in the [examples/hello_world](#) folder.

Build and run

Build and run the application.

```
$ cd examples/hello_world
$ make -s BOARD=<board> run
...
Hello world!
$
```


1.5.7 HTTP Client

About

Conenct to a remote host perform a HTTP GET action to fetch the root page '/' from the remote host.

Define CONFIG_START_NETWORK_INTERFACE_WIFI_SSID and CONFIG_START_NETWORK_INTERFACE_WIFI_PASSWORD in config.h to the SSID and password of your WiFi, otherwise the board will fail to connect to the WiFi network. Alternatively, the defines can be given as defines on the make command line as seen in the example below.

Source code

```
/**
 * @section License
 *
 * The MIT License (MIT)
 *
 * Copyright (c) 2014-2016, Erik Moqvist
 *
 * 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.
 *
 * This file is part of the Simba project.
 */

#include "simba.h"

/* The ip address of the host to connect to. */
#define REMOTE_HOST_IP 216.58.211.142

int main()
{
    struct socket_t socket;
    char http_request[] =
        "GET / HTTP/1.1\r\n"
        "Host: " STRINGIFY(REMOTE_HOST_IP) "\r\n"
        "\r\n";
    char http_response[64];
    char remote_host_ip[] = STRINGIFY(REMOTE_HOST_IP);
    struct inet_addr_t remote_host_address;
```

```

/* Start the system. Brings up the configured network interfaces
   and starts the TCP/IP-stack. */
sys_start();

/* Open the tcp socket. */
socket_open_tcp(&socket);

std_printf(FSTR("Connecting to '%s'.\r\n"), remote_host_ip);

if (inet_aton(remote_host_ip, &remote_host_address.ip) != 0) {
    std_printf(FSTR("Bad ip address '%s'.\r\n"), remote_host_ip);
    return (-1);
}

remote_host_address.port = 80;

if (socket_connect(&socket, &remote_host_address) != 0) {
    std_printf(FSTR("Failed to connect to '%s'.\r\n"), remote_host_ip);
    return (-1);
}

/* Send the HTTP request... */
if (socket_write(&socket,
                http_request,
                strlen(http_request)) != strlen(http_request)) {
    std_printf(FSTR("Failed to send the HTTP request.\r\n"));
    return (-1);
}

/* ...and receive the first 64 bytes of the response. */
if (socket_read(&socket,
                http_response,
                sizeof(http_response)) != sizeof(http_response)) {
    std_printf(FSTR("Failed to receive the response.\r\n"));
}

std_printf(FSTR("First 64 bytes of the response:\r\n"
                "%s"),
            http_response);

/* Close the socket. */
socket_close(&socket);

return (0);
}

```

The source code can also be found on Github in the [examples/http_client](#) folder.

Build and run

Build and run the application. It must be built for ESP12E or ESP01 since those are the only boards with a network connection (WiFi).

```

$ cd examples/http_client
$ make -s BOARD=esp12e CDEFS_EXTRA="CONFIG_START_NETWORK_INTERFACE_WIFI_SSID=Qvist_
↪CONFIG_START_NETWORK_INTERFACE_WIFI_PASSWORD=FooBar" run

```

```
...
Connecting to WiFi with SSID 'Qvist'.
Connected to WiFi with SSID 'Qvist'. Got IP address '192.168.1.103'.
Connecting to '216.58.211.142'.
First 64 bytes of the response:
HTTP/1.1 301 Moved Permanently
Location: http://www.google.com/GET / HTTP/1.1
Host: 216.58.211.142
...
$
```

1.5.8 Ping

About

Ping a remote host periodically once every second.

Source code

```
/**
 * @section License
 *
 * The MIT License (MIT)
 *
 * Copyright (c) 2014-2016, Erik Moqvist
 *
 * 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.
 *
 * This file is part of the Simba project.
 */

#include "simba.h"

/* The ip address of the host to ping. */
#define REMOTE_HOST_IP 216.58.211.142
```

```
int main()
{
    int res, attempt;
    char remote_host_ip[] = STRINGIFY(REMOTE_HOST_IP);
    struct inet_ip_addr_t remote_host_ip_address;
    struct time_t round_trip_time, timeout;

    sys_start();

    if (inet_aton(remote_host_ip, &remote_host_ip_address) != 0) {
        std_printf(FSTR("Bad ip address '%s'.\r\n"), remote_host_ip);
        return (-1);
    }

    timeout.seconds = 3;
    timeout.nanoseconds = 0;
    attempt = 1;

    /* Ping the remote host once every second. */
    while (1) {
        res = ping_host_by_ip_address(&remote_host_ip_address,
                                      &timeout,
                                      &round_trip_time);

        if (res == 0) {
            std_printf(FSTR("Successfully pinged '%s' (%d).\r\n"),
                      remote_host_ip,
                      attempt);
        } else {
            std_printf(FSTR("Failed to ping '%s' (%d).\r\n"),
                      remote_host_ip,
                      attempt);
        }

        attempt++;
        thrd_sleep(1);
    }

    return (0);
}
```

The source code can also be found on Github in the [examples/ping](#) folder.

Build and run

Build and run the application.

```
$ cd examples/ping
$ make -s BOARD=<board> run
Successfully pinged '192.168.1.100' in 20 ms (#1).
Successfully pinged '192.168.1.100' in 20 ms (#2).
Successfully pinged '192.168.1.100' in 20 ms (#3).
```

1.5.9 Queue

About

Use a queue to communicate between two threads.

Source code

```
/**
 * @section License
 *
 * The MIT License (MIT)
 *
 * Copyright (c) 2014-2016, Erik Moqvist
 *
 * 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.
 *
 * This file is part of the Simba project.
 */

#include "simba.h"

static struct queue_t queue;

static THRD_STACK(writer_stack, 256);

static void *writer_main(void *arg_p)
{
    int value;

    /* Write to the queue. */
    value = 1;
    queue_write(&queue, &value, sizeof(value));

    return (NULL);
}

int main()
{

```

```
int value;

sys_start();
queue_init(&queue, NULL, 0);
thrd_spawn(writer_main, NULL, 0, writer_stack, sizeof(writer_stack));

/* Read from the queue. */
queue_read(&queue, &value, sizeof(value));

std_printf(FSTR("read value = %d\r\n"), value);

return (0);
}
```

The source code can also be found on Github in the [examples/queue](#) folder.

Build and run

Build and upload the application.

```
$ cd examples/queue
$ make -s BOARD=<board> run
read value = 1
```

1.5.10 Shell

About

Use the serial port to monitor and control the application.

Source code

```
/**
 * @section License
 *
 * The MIT License (MIT)
 *
 * Copyright (c) 2014-2016, Erik Moqvist
 *
 * 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.
*
* This file is part of the Simba project.
*/

#include "simba.h"

/* Hello world command. */
static struct fs_command_t cmd_hello_world;

static struct shell_t shell;

/**
 * The shell command callback for "/hello_world".
 */
static int cmd_hello_world_cb(int argc,
                              const char *argv[],
                              void *out_p,
                              void *in_p,
                              void *arg_p,
                              void *call_arg_p)
{
    /* Write "Hello World!" to the output channel. */
    std_fprintf(out_p, FSTR("Hello World!\r\n"));

    return (0);
}

int main()
{
    /* Start the system. */
    sys_start();

    std_printf(sys_get_info());

#ifdef (__DRIVERS_I2C_H__)
    i2c_module_init();
#endif

    pin_module_init();

    /* Register the hello world command. */
    fs_command_init(&cmd_hello_world,
                   FSTR("/hello_world"),
                   cmd_hello_world_cb,
                   NULL);
    fs_command_register(&cmd_hello_world);

    /* Start the shell. */
    shell_init(&shell,
              sys_get_stdin(),
              sys_get_stdout(),
              NULL,
              NULL,

```

```
        NULL,  
        NULL);  
    shell_main(&shell);  
  
    return (0);  
}
```

The source code can also be found on Github in the [examples/shell](#) folder.

Build and run

Build and run the application.

```
$ cd examples/shell  
$ make -s BOARD=<board> upload
```

Communicate with the board using a serial terminal emulator, for example *TeraTerm*.

Type `hello_world` in the terminal emulator and press Enter. `Hello World!` is printed.

Press Tab to print a list of all registered commands and try them if you want to.

```
$ hello_world  
Hello World!  
$ <tab>  
drivers/  
filesystems/  
hello_world  
help  
history  
kernel/  
logout  
oam/  
$ kernel/thrd/list  


| NAME  | STATE   | PRIO | CPU | MAX-STACK-USAGE | LOGMASK |
|-------|---------|------|-----|-----------------|---------|
| shell | current | 0    | 0%  | 358/ 5575       | 0x0f    |
| idle  | ready   | 127  | 0%  | 57/ 156         | 0x0f    |

  
$
```

1.5.11 Timer

About

Start a periodic timer that writes an event to the main thread. The main thread reads the event and prints “timeout” to the standard output.

Source code

```
/**  
 * @section License  
 *  
 * The MIT License (MIT)  
 *  
 * Copyright (c) 2014-2016, Erik Moqvist
```



```

*
* 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.
*
* This file is part of the Simba project.
*/

#include "simba.h"

#define TIMEOUT_EVENT    0x1

static struct event_t event;
static struct timer_t timer;

static void timer_cb(void *arg_p)
{
    uint32_t mask;

    mask = TIMEOUT_EVENT;
    event_write_isr(&event, &mask, sizeof(mask));
}

int main()
{
    uint32_t mask;
    struct time_t timeout;

    sys_start();
    event_init(&event);

    /* Initialize and start a periodic timer. */
    timeout.seconds = 1;
    timeout.nanoseconds = 0;
    timer_init(&timer, &timeout, timer_cb, NULL, TIMER_PERIODIC);
    timer_start(&timer);

    while (1) {
        mask = TIMEOUT_EVENT;
        event_read(&event, &mask, sizeof(mask));

        std_printf(FSTR("timeout\\r\\n"));
    }
}

```

```
}  
  
    return (0);  
}
```

The source code can also be found on Github in the [examples/timer](#) folder.

Build and run

Build and upload the application.

```
$ cd examples/timer  
$ make -s BOARD=<board> run  
timeout  
timeout  
timeout
```

1.6 Library Reference

Simba's standard library is very extensive, offering a wide range of facilities as indicated by the long table of contents listed below. The library contains modules used by many developers in their everyday programming.

Besides the generated documentation, the source code of the interfaces and their implementatins are available on [Github](#).

1.6.1 kernel

The kernel package is the heart in *Simba*. It implements the thread scheduler.

The kernel package on [Github](#).

errno — Error numbers

Source code: [src/kernel/errno.h](#)

Defines

EPERM

ENOENT

No such file or directory.

ESRCH

No such process.

EINTR

Interrupted system call.

EIO

I/O error.

ENXIO

No such device or address.

E2BIG

Argument list too long.

ENOEXEC

Exec format error.

EBADF

Bad file number.

ECHILD

No child processes.

EAGAIN

Try again.

ENOMEM

Out of memory.

EACCES

Permission denied.

EFAULT

Bad address.

ENOTBLK

Block device required.

EBUSY

Device or resource busy.

EEXIST

File exists.

EXDEV

Cross-device link.

ENODEV

No such device.

ENOTDIR

Not a directory.

EISDIR

Is a directory.

EINVAL

Invalid argument.

ENFILE

File table overflow.

EMFILE

Too many open files.

ENOTTY

Not a typewriter.

ETXTBSY

Text file busy.

EFBIG

File too large.

ENOSPC

No space left on device.

ESPIPE

Illegal seek.

EROFS

Read-only file system.

EMLINK

Too many links.

EPIPE

Broken pipe.

EDOM

Math argument out of domain of func.

ERANGE

Math result not representable.

EDEADLK

Resource deadlock would occur.

ENAMETOOLONG

File name too long.

ENOLCK

No record locks available.

ENOSYS

Function not implemented.

ENOTEMPTY

Directory not empty.

ELOOP

Too many symbolic links encountered.

EWouldBlock

Operation would block.

ENOMSG

No message of desired type.

EIDRM

Identifier removed.

ECHRNG

Channel number out of range.

EL2NSYNC

Level 2 not synchronized.

EL3HLT

Level 3 halted.

EL3RST

Level 3 reset.

ELNRNG

Link number out of range.

EUNATCH

Protocol driver not attached.

ENOC SI

No CSI structure available.

EL2HLT

Level 2 halted.

EBADE

Invalid exchange.

EBADR

Invalid request descriptor.

EXFULL

Exchange full.

ENOANO

No anode.

EBADRQC

Invalid request code.

EBADSLT

Invalid slot.

EDEADLOCK**EBFONT**

Bad font file format.

ENOSTR

Device not a stream.

ENODATA

No data available.

ETIME

Timer expired.

ENOSR

Out of streams resources.

ENONET

Machine is not on the network.

ENOPKG

Package not installed.

EREMOTE

Object is remote.

ENOLINK

Link has been severed.

EADV

Advertise error.

ESRMNT

Srmount error.

ECOMM

Communication error on send.

EPROTO

Protocol error.

EMULTIHOP

Multihop attempted.

EDOTDOT

RFS specific error.

EBADMSG

Not a data message.

EOVERFLOW

Value too large for defined data type.

ENOTUNIQ

Name not unique on network.

EBADFD

File descriptor in bad state.

EREMCHG

Remote address changed.

ELIBACC

Can not access a needed shared library.

ELIBBAD

Accessing a corrupted shared library.

ELIBSCN

.lib section in a.out corrupted.

ELIBMAX

Attempting to link in too many shared libraries.

ELIBEXEC

Cannot exec a shared library directly.

EILSEQ

Illegal byte sequence.

ERESTART

Interrupted system call should be restarted.

ESTRPIPE

Streams pipe error.

EUSERS

Too many users.

ENOTSOCK

Socket operation on non-socket.

EDESTADDRREQ

Destination address required.

EMSGSIZE

Message too long.

EPROTOTYPE

Protocol wrong type for socket.

ENOPROTOOPT

Protocol not available.

EPROTONOSUPBOARD

Protocol not supported.

ESOCKTNOSUPBOARD

Socket type not supported.

EOPNOTSUPP

Operation not supported on transport endpoint.

EPFNOSUPBOARD

Protocol family not supported.

EAFNOSUPBOARD

Address family not supported by protocol.

EADDRINUSE

Address already in use.

EADDRNOTAVAIL

Cannot assign requested address.

ENETDOWN

Network is down.

ENETUNREACH

Network is unreachable.

ENETRESET

Network dropped connection because of reset.

ECONNABORTED

Software caused connection abort.

ECONNRESET

Connection reset by peer.

ENOBUFS

No buffer space available.

EISCONN

Transport endpoint is already connected.

ENOTCONN

Transport endpoint is not connected.

ESHUTDOWN

Cannot send after transport endpoint shutdown.

ETOOMANYREFS

Too many references: cannot splice.

ETIMEDOUT

Connection timed out.

ECONNREFUSED

Connection refused.

EHOSTDOWN

Host is down.

EHOSTUNREACH

No route to host.

EALREADY

Operation already in progress.

EINPROGRESS

Operation now in progress.

ESTALE

Stale NFS file handle.

EUCLEAN

Structure needs cleaning.

ENOTNAM

Not a XENIX named type file.

ENAVAIL

No XENIX sems available.

EISNAM

Is a named type file.

EREMOTEIO

Remote I/O error.

EDQUOT

Quota exceeded.

ENOMEDIUM

No medium found.

EMEDIUMTYPE

Wrong medium type.

ECANCELED

Operation Canceled.

ENOKEY

Required key not available.

EKEYEXPIRED

Key has expired.

EKEYREVOKED

Key has been revoked.

EKEYREJECTED

Key was rejected by service.

ESTACK

Stack corrupt.

EBTASSERT

Test assertion.

sys — System

System level functionality and definitions.

Source code: `src/kernel/sys.h`, `src/kernel/sys.c`

Test code: `tst/kernel/sys/main.c`

Test coverage: `src/kernel/sys.c`

Defines

VERSION_STR

SYS_TICK_MAX

Typedefs

typedef uint64_t **sys_tick_t**

Functions

static *sys_tick_t* **t2st** (*struct time_t* *time_p)

Conversion from the time struct to system ticks.

static void **st2t** (*sys_tick_t* tick, *struct time_t* *time_p)

Conversion from system ticks to the time struct.

int **sys_module_init** (void)

Initialize the sys module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

int **sys_start** (void)

Start the system and convert this context to the main thread.

This function initializes a bunch of enabled features in the simba platform. Many low level features (scheduling, timers, ...) are always enabled, but higher level features are only enabled if configured.

This function **must** be the first function call in `main()`.

Return zero(0) or negative error code.

void **sys_stop** (int error)

Stop the system.

Return Never returns.

void **sys_reboot** (void)

Reboot the system. Sets all registers to their known, default values and restarts the application. Also known as a soft reset.

Return Never returns.

void **sys_set_on_fatal_callback** (void (**callback*)) int error

Set the on-fatal-callback function to given callback.

The on-fatal-callback is called when a fatal error occurs. The default on-fatal-callback is `sys_stop()`.

Return void

Parameters

- *callback*: Callback called when a fatal error occurs.

void **sys_set_stdin** (void **chan_p*)

Set the standard input channel.

Return void.

Parameters

- *chan_p*: New standard input channel.

void ***sys_get_stdin** (void)

Get the standard input channel.

Return Standard input channel or NULL.

void **sys_set_stdout** (void **chan_p*)

Set the standard output channel.

Return void.

Parameters

- *chan_p*: New standard output channel.

void ***sys_get_stdout** (void)

Get the standard output channel.

Return Standard output channel or NULL.

void **sys_lock** (void)

Take the system lock. Turns off interrupts.

Return void.

void **sys_unlock** (void)

Release the system lock. Turn on interrupts.

Return void.

void **sys_lock_isr** (void)

Take the system lock from isr. In many ports this has no effect.

Return void.

void **sys_unlock_isr** (void)

Release the system lock from isr. In many ports this function has no effect.

Return void.

far_string_t **sys_get_info** (void)

Get a pointer to the application information buffer.

The buffer contains various information about the application; for example the application name and the build date.

Return The pointer to the application information buffer.

far_string_t **sys_get_config** (void)

Get a pointer to the application configuration buffer.

The buffer contains a string of all configuration variables and their values.

Return The pointer to the application configuration buffer.

float **sys_interrupt_cpu_usage_get** (void)

Get the current interrupt cpu usage counter.

Return cpu usage, 0-100.

void **sys_interrupt_cpu_usage_reset** (void)

Reset the interrupt cpu usage counter.

Variables

struct *sys_t* **sys**

struct **sys_t**

Public Members

sys_tick_t **tick**

void (***on_fatal_callback**) (int error)

void ***stdin_p**

void ***stdout_p**

uint32_t **start**

uint32_t **time**

struct sys_t::@63 sys_t::interrupt

thrd — Threads

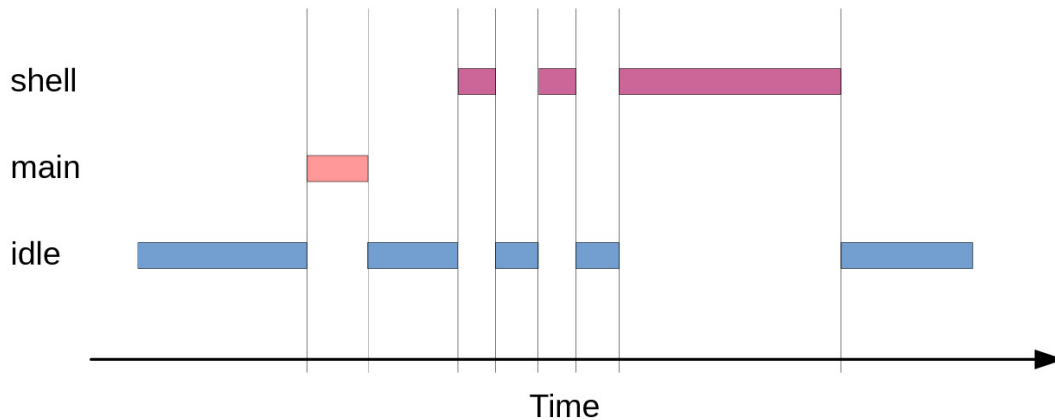
A thread is the basic execution entity in the OS. A pre-emptive or cooperative scheduler controls the execution of threads.

Scheduler

The single core scheduler is configured as cooperative or preemptive at compile time. The cooperative scheduler is implemented for all boards, but the preemptive scheduler is only implemented for a few boards.

There are two threads that are always present; the main thread and the idle thread. The main thread is the root thread in the system, created in the `main()` function by calling `sys_start()`. The idle thread is running when no other thread is ready to run. It simply waits for an interrupt to occur and then reschedules to run other ready threads.

The diagram below is an example of how three threads; `shell`, `main` and `idle` are scheduled over time.



As it is a single core scheduler only one thread is running at a time. In the beginning the system is idle and the `idle` thread is running. After a while the `main` and `shell` threads have some work to do, and since they have higher priority than the `idle` thread they are scheduled. At the end the `idle` thread is running again.

Debug file system commands

Four debug file system commands are available, all located in the directory `kernel/thrd/`.

Command	Description
<code>list</code>	Print a list of all threads.
<code>set_log_mask <thread name> <mask></code>	Set the log mask of thread <code><thread name></code> to <code>mask</code> .
<code>monitor/set_period_ms <ms></code>	Set the monitor thread sampling period to <code><ms></code> milliseconds.
<code>monitor/set_print <state></code>	Enable(1)/disable(0) monitor statistics to be printed periodically.

Example output from the shell:

```
$ kernel/thrd/list
```

NAME	STATE	PRIO	CPU	SCHEDULED	LOGMASK
main	current	0	0%	1	0x0f
	ready	127	0%	0	0x0f
	ready	-80	0%	0	0x0f

Source code: [src/kernel/thrd.h](#), [src/kernel/thrd.c](#)

Test code: [tst/kernel/thrd/main.c](#)

Test coverage: [src/kernel/thrd.c](#)

Defines

THRD_STACK (name, size)

THRD_CONTEXT_STORE_ISR

Push all callee-save registers not part of the context struct. The preemptive scheduler requires this macro before the `thrd_yield_isr()` function is called from interrupt context.

THRD_CONTEXT_LOAD_ISR

Pop all callee-save registers not part of the context struct. The preemptive scheduler requires this macro after the `thrd_yield_isr()` function is called from interrupt context.

THRD_RESCHEDULE_ISR

Reschedule from isr. Used by preemptive systems to interrupt low priority threads in favour of high priority threads.

Functions

int **thrd_module_init** (void)

Initialize the thread module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code

struct *thrd_t* ***thrd_spawn** (void *(*main)) void *

, void *arg_p, int prio, void *stack_p, size_t stack_size Spawn a thread with given main (entry) function and argument. The thread is initialized and added to the ready queue in the scheduler for execution when prioritized.

Return Thread id, or NULL on error.

Parameters

- `main`: Thread main (entry) function. This function normally contains an infinite loop waiting for events to occur.
- `arg_p`: Main function argument. Passed as `arg_p` to the main function.
- `prio`: Thread scheduling priority. [-127..127], where -127 is the highest priority and 127 is the lowest.
- `stack_p`: Stack pointer. The pointer to a stack created with the macro `THRD_STACK()`.
- `stack_size`: The stack size in number of bytes.

int **thrd_suspend** (struct *time_t* *timeout_p)

Suspend current thread and wait to be resumed or a timeout occurs (if given).

Return zero(0), -ETIMEOUT on timeout or other negative error code.

Parameters

- `timeout_p`: Time to wait to be resumed before a timeout occurs and the function returns.

int **thrd_resume** (struct *thrd_t* *thrd_p, int err)

Resume given thread. If resumed thread is not yet suspended it will not be suspended on next suspend call to `thrd_suspend()` or `thrd_suspend_isr()`.

Return zero(0) or negative error code.

Parameters

- `thrd_p`: Thread id to resume.
- `err`: Error code to be returned by `thrd_suspend()` or `thrd_suspend_isr()`.

int **thrd_yield** (void)

Put the currently executing thread on the ready list and reschedule.

This function is often called periodically from low priority work heavy threads to give higher priority threads the chance to execute.

Return zero(0) or negative error code.

int **thrd_join** (struct *thrd_t* **thrd_p*)

Wait for given thread to terminate.

Return zero(0) or negative error code.

Parameters

- `thrd_p`: Thread to wait for.

int **thrd_sleep** (float *seconds*)

Pauses the current thread for given number of seconds.

Return zero(0) or negative error code.

Parameters

- `seconds`: Seconds to sleep.

int **thrd_sleep_ms** (int *ms*)

Pauses the current thread for given number of milliseconds.

Return zero(0) or negative error code.

Parameters

- `ms`: Milliseconds to sleep.

int **thrd_sleep_us** (long *us*)

Pauses the current thread for given number of microseconds.

Return zero(0) or negative error code.

Parameters

- `us`: Microseconds to sleep.

struct *thrd_t* ***thrd_self** (void)

Get current thread's id.

Return Thread id.

int **thrd_set_name** (const char **name_p*)

Set the name of the current thread.

Return zero(0) or negative error code.

Parameters

- `name_p`: New thread name.

const char *`thrd_get_name` (void)
Get the name of the current thread.

Return Current thread name.

struct *thrd_t* *`thrd_get_by_name` (const char **name_p*)
Get the pointer to given thread.

Return Thraed pointer or NULL if the thread was not found.

int `thrd_set_log_mask` (struct *thrd_t* **thrd_p*, int *mask*)
Set the log mask of given thread.

Return Old log mask.

Parameters

- `thrd_p`: Thread to set the log mask of.
- `mask`: Log mask. See the log module for available levels.

int `thrd_get_log_mask` (void)
Get the log mask of the current thread.

Return Log mask of current thread.

int `thrd_set_prio` (struct *thrd_t* **thrd_p*, int *prio*)
Set the priority of given thread.

Return zero(0) or negative error code.

Parameters

- `thrd_p`: Thread to set the priority for.
- `prio`: Priority.

int `thrd_get_prio` (void)
Get the priority of the current thread.

Return Priority of current thread.

int `thrd_init_global_env` (struct *thrd_environment_variable_t* **variables_p*, int *length*)
Initialize the global environment variables storage. These variables are shared among all threads.

Return zero(0) or negative error code.

Parameters

- `variables_p`: Variables array.
- `length`: Length of the variables array.

int **thrd_set_global_env** (const char **name_p*, const char **value_p*)

Set the value of given environment variable. The pointers to given name and value are stored in the current global environment array.

Return zero(0) or negative error code.

Parameters

- *name_p*: Name of the environment variable to set.
- *value_p*: Value of the environment variable. Set to NULL to remove the variable.

const char ***thrd_get_global_env** (const char **name_p*)

Get the value of given environment variable in the global environment array.

Return Value of given environment variable or NULL if it is not found.

Parameters

- *name_p*: Name of the environment variable to get.

int **thrd_init_env** (struct *thrd_environment_variable_t* **variables_p*, int *length*)

Initialize the current threads' environment variables storage.

Return zero(0) or negative error code.

Parameters

- *variables_p*: Variables are to be used by this thread.
- *length*: Length of the variables array.

int **thrd_set_env** (const char **name_p*, const char **value_p*)

Set the value of given environment variable. The pointers to given name and value are stored in the current threads' environment array.

Return zero(0) or negative error code.

Parameters

- *name_p*: Name of the environment variable to set.
- *value_p*: Value of the environment variable. Set to NULL to remove the variable.

const char ***thrd_get_env** (const char **name_p*)

Get the value of given environment variable. If given variable is not found in the current threads' environment array, the global environment array is searched.

Return Value of given environment variable or NULL if it is not found.

Parameters

- *name_p*: Name of the environment variable to get.

int **thrd_suspend_isr** (struct *time_t* **timeout_p*)

Suspend current thread with the system lock taken (see `sys_lock()`) and wait to be resumed or a timeout occurs (if given).

Return zero(0), -ETIMEDOUT on timeout or other negative error code.

Parameters

- `timeout_p`: Time to wait to be resumed before a timeout occurs and the function returns.

`int thrd_resume_isr (struct thrd_t *thrd_p, int err)`

Resume given thread from isr or with the system lock taken (see `sys_lock()`). If resumed thread is not yet suspended it will not be suspended on next suspend call to `thrd_suspend()` or `thrd_suspend_isr()`.

Return zero(0) or negative error code.

Parameters

- `thrd_p`: Thread id to resume.
- `err`: Error code to be returned by `thrd_suspend()` or `thrd_suspend_isr()`.

`int thrd_yield_isr (void)`

Yield current thread from isr (preemptive scheduler only) or with the system lock taken.

Return zero(0) or negative error code.

`struct thrd_environment_variable_t`

#include <thrd.h> A thread environment variable.

Public Members

`const char *name_p`

`const char *value_p`

`struct thrd_environment_t`

Public Members

`struct thrd_environment_variable_t *variables_p`

`size_t number_of_variables`

`size_t max_number_of_variables`

`struct thrd_t`

Public Members

`struct thrd_t *prev_p`

`struct thrd_t *next_p`

`struct thrd_t::@64 thrd_t::scheduler`

`struct thrd_port_t port`

`int prio`

`int state`

`int err`

`int log_mask`

`struct timer_t *timer_p`

```
const char *name_p
struct thrd_t::@65  thrd_t::statistics
```

time — System time

Source code: [src/kernel/time.h](#), [src/kernel/time.c](#)

Test code: [tst/kernel/time/main.c](#)

Test coverage: [src/kernel/time.c](#)

Functions

int **time_get** (struct *time_t* *now_p)

Get current time in seconds and nanoseconds. The resolution of the time is implementation specific and may vary a lot between different architectures.

Return zero(0) or negative error code.

Parameters

- now_p: Read current time.

int **time_set** (struct *time_t* *new_p)

Set current time in seconds and nanoseconds.

Return zero(0) or negative error code.

Parameters

- new_p: New current time.

int **time_diff** (struct *time_t* *diff_p, struct *time_t* *left_p, struct *time_t* *right_p)

Subtract given times.

Return zero(0) or negative error code.

Parameters

- diff_p: The result of the subtranting left_p from right_p.
- left_p: The operand to subtract from.
- right_p: The operand to subtract.

void **time_busy_wait_us** (long *useconds*)

Busy wait for given number of microseconds.

NOTE: The maximum allowed time to sleep is target specific.

Return void

Parameters

- useconds: Microseconds to sleep.

`int time_unix_time_to_date (struct date_t *date_p, struct time_t *time_p)`
 Convert given unix time to a date.

Return zero(0) or negative error code.

Parameters

- `date_p`: Converted time.
- `time_p`: Unix time to convert.

`struct time_t`
`#include <time.h>`

Public Members

`int32_t seconds`
 Number of seconds.

`int32_t nanoseconds`
 Number of nanoseconds.

`struct date_t`
`#include <time.h>` A date in year, month, date, day, hour, minute and seconds.

Public Members

`int second`
 Second [0..59].

`int minute`
 Minute [0..59].

`int hour`
 Hour [0..23].

`int day`
 Weekday [1..7], where 1 is Monday and 7 is Sunday.

`int date`
 Day in month [1..31]

`int month`
 Month [1..12] where 1 is January and 12 is December.

`int year`
 Year [1970..].

timer — Timers

Timers are started with a timeout, and when the time is up the timer expires and the timer callback function is called from interrupt context.

The timeout resolution is the system tick period. Timeouts are always rounded up to the closest system tick. That is, a timer can never expire early, but may expire slightly late.

An application requiring timers with higher precision than the system tick must use the hardware timers.

Source code: [src/kernel/timer.h](#), [src/kernel/timer.c](#)

Test code: [tst/kernel/timer/main.c](#)

Test coverage: [src/kernel/timer.c](#)

Defines

TIMER_PERIODIC

Typedefs

typedef void (***timer_callback_t**) (void *arg_p)
Time callback prototype.

Functions

int timer_module_init (void)
Initialize the timer module. This function must be called before calling any other function in this module.
The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

int timer_init (**struct timer_t** *self_p, **struct time_t** *timeout_p, **timer_callback_t** callback, void *arg_p, int flags)
Initialize given timer object with given timeout and expiry callback. The timer resolution directly depends on the system tick frequency and is rounded up to the closest possible value. This applies to both single shot and periodic timers.

Return zero(0) or negative error code.

Parameters

- self_p: Timer object to initialize with given parameters.
- timeout_p: The timer timeout value.
- callback: Function called when the timer expires. Called from interrupt context.
- arg_p: Function callback argument. Passed to the callback when the timer expires.
- flags: Set **TIMER_PERIODIC** for periodic timer.

int timer_start (**struct timer_t** *self_p)
Start given initialized timer object.

Return zero(0) or negative error code.

Parameters

- self_p: Timer object to start.

int **timer_start_isr** (struct *timer_t* *self_p)
 See `timer_start()` for a description.

This function may only be called from an isr or with the system lock taken (see `sys_lock()`).

int **timer_stop** (struct *timer_t* *self_p)
 Stop given timer object. This has no effect on a timer that already expired or was never started. The return code is 0 if the timer was stopped and -1 otherwise.

Return zero(0) if the timer was stopped and -1 if the timer has already expired or was never started.

Parameters

- self_p: Timer object to stop.

int **timer_stop_isr** (struct *timer_t* *self_p)
 See `timer_stop()` for description.

This function may only be called from an isr or with the system lock taken (see `sys_lock()`).

struct **timer_t**

Public Members

```
struct timer_t *next_p
sys_tick_t delta
sys_tick_t timeout
int flags
timer_callback_t callback
void *arg_p
```

types — Common types

Source code: [src/kernel/types.h](#)

Defines

UNUSED (v)
STRINGIFY (x)
 Create a string of an identifier using the pre-processor.
STRINGIFY2 (x)
 Used internally by `STRINGIFY()`.
TOKENPASTE (x, y)
 Concatenate two tokens.
TOKENPASTE2 (x, y)
 Used internally by `TOKENPASTE()`.
UNIQUE (x)
 Create a unique token.

membersof (a)

Get the number of elements in an array.

As an example, the code below outputs number of members in foo = 10.

```
int foo[10];

std_printf(FSTR("number of members in foo = %d\\r\\n"),
           membersof(foo));
```

container_of (ptr, type, member)**DIV_CEIL** (n, d)

Integer division that rounds the result up.

MIN (a, b)

Get the minimum value of the two.

MAX (a, b)

Get the maximum value of the two.

PRINT_FILE_LINE

Debug print of file and line.

STD_PRINTF_DEBUG (...)**_ASSERTFMT** (fmt, ...)**ASSERTN** (cond, n, ...)

Assert given condition and call the system on fatal callback with given value n on error.

ASSERT (cond, ...)

Assert given condition and call the system on fatal callback with value 1 on error.

BIT (pos)**BITFIELD_SET** (name, value)**BITFIELD_GET** (name, value)

1.6.2 drivers

The drivers package on [Github](#).

Modules:

adc — Analog to digital conversion

Source code: [src/drivers/adc.h](#), [src/drivers/adc.c](#)

Test code: [tst/drivers/adc/main.c](#)

Defines

ADC_REFERENCE_VCC

Functions

int **adc_module_init** (void)

Initialize the ADC driver module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

int **adc_init** (struct adc_driver_t *self_p, struct adc_device_t *dev_p, struct pin_device_t *pin_dev_p, int reference, long sampling_rate)

Initialize given driver object from given configuration.

Return zero(0) or negative error code.

Parameters

- self_p: Driver object to be initialized.
- dev_p: ADC device to use.
- pin_dev_p: Pin device to use.
- reference: Voltage reference. Only ADC_REFERENCE_VCC is supported.
- sampling_rate: Sampling rate in Hz. The lowest allowed value is one and the highest value depends on the architecture. The sampling rate is not used in single sample conversions, ie. calls to `adc_async_convert()` and `adc_convert()` with length one; or calls to `adc_convert_isr()`.

int **adc_async_convert** (struct adc_driver_t *self_p, uint16_t *samples_p, size_t length)

Start an asynchronous conversion of analog signal to digital samples. Call `adc_async_wait()` to wait for the conversion to complete.

Return zero(0) or negative error code.

Parameters

- self_p: Driver object.
- samples_p: Converted samples.
- length: Length of samples array.

int **adc_async_wait** (struct adc_driver_t *self_p)

Wait for an asynchronous conversion to complete.

Return zero(0) or negative error code.

Parameters

- self_p: Driver object.

int **adc_convert** (struct adc_driver_t *self_p, uint16_t *samples_p, size_t length)

Start a synchronous conversion of an analog signal to digital samples. This is equivalent to `adc_async_convert() + adc_async_wait()`, but in a single function call.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object.
- `samples_p`: Converted samples.
- `length`: Length of samples array.

int **adc_convert_isr** (**struct** `adc_driver_t` **self_p*, uint16_t **sample_p*)

Start a synchronous conversion of analog signal to digital samples from isr or with the system lock taken. This function will poll the ADC hardware until the sample has been converted.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object.
- `sample_p`: Converted sample.

Variables

struct `adc_device_t` **adc_device**[ADC_DEVICE_MAX]

`analog_input_pin` — Analog input pin

Source code: `src/drivers/analog_input_pin.h`, `src/drivers/analog_input_pin.c`

Test code: `tst/drivers/analog_input_pin/main.c`

Functions

int **analog_input_pin_module_init** (void)

Initialize the analog input pin module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

int **analog_input_pin_init** (**struct** *analog_input_pin_t* **self_p*, **struct** `pin_device_t` **dev_p*)

Initialize given driver object with given device and mode.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object to be initialized.
- `dev_p`: Device to use.

int **analog_input_pin_read** (**struct** *analog_input_pin_t* **self_p*)

Read the current value of given pin.

Return Analog pin value, otherwise negative error code.

Parameters

- `self_p`: Driver object.

int **analog_input_pin_read_isr** (struct *analog_input_pin_t* **self_p*)
Read the current value of given pin from an isr or with the system lock taken.

Return Analog pin value, otherwise negative error code.

Parameters

- `self_p`: Driver object.

struct **analog_input_pin_t**
#include <analog_input_pin.h>

Public Members

struct **adc_driver_t** **adc**

analog_output_pin — Analog output pin

Source code: [src/drivers/analog_output_pin.h](#), [src/drivers/analog_output_pin.c](#)

Test code: [tst/drivers/analog_output_pin/main.c](#)

Functions

int **analog_output_pin_module_init** (void)
Initialize the analog output pin module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

int **analog_output_pin_init** (struct *analog_output_pin_t* **self_p*, struct *pin_device_t* **dev_p*)
Initialize given driver object with given device and mode.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object to be initialized.
- `dev_p`: Device to use.

int **analog_output_pin_write** (struct *analog_output_pin_t* **self_p*, int *value*)
Write given value to the analog pin.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object.

- **value**: The value to write to the pin. A number in the range 0 to 1023, where 0 is lowest output and 1023 is highest output.

int **analog_output_pin_read** (struct *analog_output_pin_t* *self_p)
Read the value that is currently written to given analog output pin.

Return Value in the range 0 to 1023, or negative error code.

Parameters

- **self_p**: Driver object.

struct **analog_output_pin_t**
#include <analog_output_pin.h>

Public Members

struct pwm_driver_t **pwm**

bcm43362 — BCM43362

BCM43362 is a WiFi module by Boadcom.

Homepage: <https://www.broadcom.com/products/wireless-connectivity/wireless-lan/bcm43362>

Source code: [src/drivers/bcm43362.h](#), [src/drivers/bcm43362.c](#)

Test code: [tst/drivers/bcm43362/main.c](#)

Functions

int **bcm43362_module_init** (void)
Initialize the BCM43362 module. This function must be called before calling any other function in this module.
The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

int **bcm43362_init** (struct *bcm43362_driver_t* *self_p, struct sdio_device_t *sdio_dev_p)
Initialize driver object from given configuration.

Return zero(0) or negative error code.

Parameters

- **self_p**: Driver object to be initialized.
- **sdio_dev_p**: SDIO device to use.

int **bcm43362_start** (struct *bcm43362_driver_t* *self_p)
Starts the BCM43362 device using given driver object.

After a successful start of the device the application may call `bcm43362_connect ()` to connect to an AP.

Return zero(0) or negative error code.

Parameters

- `self_p`: Initialized driver object.

int **bcm43362_stop** (**struct** *bcm43362_driver_t* **self_p*)
Stops the BCM43362 device referenced by given driver object.

Return zero(0) or negative error code.

Parameters

- `self_p`: Initialized driver object.

int **bcm43362_connect** (**struct** *bcm43362_driver_t* **self_p*, **const** char **ssid_p*, **const** char **password_p*)
Connect to an WiFi Access Point (AP) with given SSID and password.

Return zero(0) or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `ssid_p`: SSID of the WiFi AP to connect to.
- `password_p`: Password.

int **bcm43362_disconnect** (**struct** *bcm43362_driver_t* **self_p*)
Disconnect from any connected WiFi AP.

Return zero(0) or negative error code.

Parameters

- `self_p`: Initialized driver object.

ssize_t **bcm43362_read** (**struct** *bcm43362_driver_t* **self_p*, **struct** pbuf **pbuf_p*, size_t *size*)
Read a packet from the BCM43362 device.

Return Number of read bytes or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `pbuf_p`: Buffer to read into.
- `size`: Number of bytes to receive.

ssize_t **bcm43362_write** (**struct** *bcm43362_driver_t* **self_p*, **struct** pbuf **pbuf_p*, size_t *size*)
Write given packet to the BCM43362 device to transmit it on the network.

This function is normally called by a network interface to send a frame on the network.

Return Number of written bytes or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `pbuf_p`: Buffer to write.

- `size`: Number of bytes to write.

```
struct bcm43362_driver_t
#include <bcm43362.h>
```

Public Members

```
struct sdio_driver_t sdio
```

can — CAN bus

A Controller Area Network (CAN bus) is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other in applications without a host computer. It is a message-based protocol, designed originally for multiplex electrical wiring within automobiles, but is also used in many other contexts.

Below is a short example of how to use this module. The error handling is left out for readability.

```
struct can_frame_t can_rx_buf[8];
struct can_frame_t frame;

/* Initialize and start the CAN controller. */
can_init(&can,
        &can_device[0],
        CAN_SPEED_500KBPS,
        can_rx_buf,
        sizeof(can_rx_buf)) == 0);
can_start(&can);

/* Read a frame from the bus. */
can_read(&can, &frame, sizeof(frame));

/* Stop the CAN controller. */
can_stop(&can);
```

Source code: [src/drivers/can.h](#), [src/drivers/can.c](#)

Test code: [tst/drivers/can/main.c](#)

Defines

CAN_SPEED_1000KBPS

CAN_SPEED_500KBPS

CAN_SPEED_250KBPS

Functions

```
int can_init (struct can_driver_t *self_p, struct can_device_t *dev_p, uint32_t speed, void *rxbuf_p, size_t
              size)
Initialize given driver object from given configuration.
```

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object to initialize.
- `dev_p`: CAN device to use.
- `speed`: Can bus speed. One of the defines with the prefix `CAN_SPEED_`.
- `rxbuf_p`: CAN frame reception buffer.
- `size`: Size of the reception buffer in bytes.

int **can_start** (**struct** can_driver_t **self_p*)

Starts the CAN device using configuration in given driver object.

Return zero(0) or negative error code.

Parameters

- `self_p`: Initialized driver object.

int **can_stop** (**struct** can_driver_t **self_p*)

Stops the CAN device referenced by given driver object.

Return zero(0) or negative error code.

Parameters

- `self_p`: Initialized driver object.

ssize_t **can_read** (**struct** can_driver_t **self_p*, **struct** can_frame_t **frame_p*, size_t *size*)

Read one or more CAN frames from the CAN bus. Blocks until the frame(s) are received.

Return Number of bytes read or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `frame_p`: Array of read frames.
- `size`: Size of frames buffer in bytes. Must be a multiple of `sizeof(struct can_frame_t)`.

ssize_t **can_write** (**struct** can_driver_t **self_p*, **const struct** can_frame_t **frame_p*, size_t *size*)

Write one or more CAN frames to the CAN bus. Blocks until the frame(s) have been transmitted.

Return Number of bytes written or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `frame_p`: Array of frames to write.
- `size`: Size of frames buffer in bytes. Must be a multiple of `sizeof(struct can_frame_t)`.

Variables

struct can_device_t **can_device**[CAN_DEVICE_MAX]

struct can_frame_t

Public Members

uint32_t **id**

int **extended_frame**

int **size**

int **rtr**

uint32_t **timestamp**

uint8_t **u8**[8]

uint32_t **u32**[2]

union can_frame_t::@0 can_frame_t::data

chipid — Chip identity

Source code: [src/drivers/chipid.h](#), [src/drivers/chipid.c](#)

Test code: [tst/drivers/chipid/main.c](#)

Functions

int **chipid_read** (struct chipid_t **id_p*)

dac — Digital to analog conversion

Source code: [src/drivers/dac.h](#), [src/drivers/dac.c](#)

Test code: [tst/drivers/dac/main.c](#)

Functions

int **dac_module_init** (void)

Initialize DAC driver module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

int **dac_init** (struct dac_driver_t **self_p*, struct dac_device_t **dev_p*, struct pin_device_t **pin0_dev_p*,
 struct pin_device_t **pin1_dev_p*, int *sampling_rate*)

Initialize given driver object from given configuration.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object to be initialized.
- `dev_p`: Device to use.
- `pin0_dev_p`: Pin used for mono or first stereo channel.
- `pin1_dev_p`: Second stereo pin.
- `sampling_rate`: Sampling rate in Hz.

int **`dac_async_convert`** (**`struct`** `dac_driver_t` *`self_p`, `uint32_t` *`samples_p`, `size_t` `length`)
Start an asynchronous conversion of samples to an analog signal.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object.
- `samples`: Samples to convert to an analog signal.
- `length`: Length of samples array.

int **`dac_async_wait`** (**`struct`** `dac_driver_t` *`self_p`)
Wait for ongoing asynchronous conversion to finish.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object.

int **`dac_convert`** (**`struct`** `dac_driver_t` *`self_p`, `uint32_t` *`samples_p`, `size_t` `length`)
Start synchronous conversion of samples to an analog signal.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object.
- `samples`: Converted samples.
- `length`: Length of samples array.

Variables

`struct` `dac_device_t` **`dac_device`**[DAC_DEVICE_MAX]

ds18b20 — One-wire temperature sensor

Source code: `src/drivers/ds18b20.h`, `src/drivers/ds18b20.c`

Test code: `tst/drivers/ds18b20/main.c`

Defines

DS18B20_FAMILY_CODE

Functions

int **ds18b20_module_init** (void)

Initialize the DS18B20 driver module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

int **ds18b20_init** (struct *ds18b20_driver_t* *self_p, struct *owi_driver_t* *owi_p)

Initialize given driver object. The driver object will communicate with all DS18B20 devices on given OWI bus.

Return zero(0) or negative error code.

Parameters

- self_p: Driver object to be initialized.
- owi_p: One-Wire (OWI) driver.

int **ds18b20_convert** (struct *ds18b20_driver_t* *self_p)

Start temperature conversion on all sensors.

Return zero(0) or negative error code.

Parameters

- self_p: Driver object to be initialized.

int **ds18b20_get_temperature** (struct *ds18b20_driver_t* *self_p, const uint8_t *id_p, int *temp_p)

Get the temperature for given device identity.

Return zero(0) or negative error code.

Parameters

- self_p: Driver object to be initialized.
- id_p: Device identity.
- temp_p: Measured temperature in Q4.4 to Q8.4 depending on resolution.

char ***ds18b20_get_temperature_str** (struct *ds18b20_driver_t* *self_p, const uint8_t *id_p, char *temp_p)

Get temperature for given device identity formatted as a string.

Return temp_p on success, NULL otherwise.

Parameters

- self_p: Driver object to be initialized.
- id_p: Device identity.
- temp_p: Measured formatted temperature.


```
struct ds18b20_driver_t
```

Public Members

```
struct owi_driver_t *owi_p
struct ds18b20_driver_t *next_p
```

ds3231 — RTC clock

Source code: `src/drivers/ds3231.h`, `src/drivers/ds3231.c`

Test code: `tst/drivers/ds3231/main.c`

Functions

int **ds3231_init** (struct *ds3231_driver_t* *self_p, struct *i2c_driver_t* *i2c_p)
Initialize given driver object.

Return zero(0) or negative error code.

Parameters

- self_p: Driver object to be initialized.
- i2c_p: I2C driver to use.

int **ds3231_set_date** (struct *ds3231_driver_t* *self_p, struct *date_t* *date_p)
Set date in the DS3231 device.

Return zero(0) or negative error code.

Parameters

- self_p: Driver object.
- date_p: Date to set in the device.

int **ds3231_get_date** (struct *ds3231_driver_t* *self_p, struct *date_t* *date_p)
Get date from the DS3231 device.

Return zero(0) or negative error code.

Parameters

- self_p: Driver object.
- date_p: Date read from the device.

```
struct ds3231_driver_t
#include <ds3231.h>
```

Public Members

```
struct i2c_driver_t *i2c_p
```

esp_wifi — Espressif WiFi

This module is a wrapper for the Espressif WiFi interface.

Configure the WiFi as a Station and an Access Point at the same time. The application tries to connect to a Wifi with SSID *ssid* and will accept connections to the SSID *Simba*.

```
esp_wifi_set_op_mode(esp_wifi_op_mode_station_softap_t);
esp_wifi_softap_init("Simba", NULL);
esp_wifi_station_init("ssid", "password", NULL);
```

Configure the WiFi as an Access Point. The application will accept connections to the SSID *Simba*.

```
esp_wifi_set_op_mode(esp_wifi_op_mode_softap_t);
esp_wifi_softap_init("Simba", NULL);
```

Configure the WiFi as a Station. The application tries to connect to a Wifi with SSID *ssid*.

```
esp_wifi_set_op_mode(esp_wifi_op_mode_station_t);
esp_wifi_station_init("ssid", "password", NULL);
```

Submodules:

esp_wifi_softap — Espressif WiFi SoftAP

This module is a wrapper for the Espressif WiFi SoftAP interface.

Source code: [src/drivers/esp_wifi/softap.h](#), [src/drivers/esp_wifi/softap.c](#)

Test code: [tst/drivers/esp_wifi/softap/main.c](#)

Functions

int **esp_wifi_softap_init** (const char *ssid_p, const char *password_p)
Initialize the WiFi SoftAP interface.

Return zero(0) or negative error code.

Parameters

- ssid_p: SSID of the SoftAP.
- password_p: Password of SoftAP.

int **esp_wifi_softap_set_ip_info** (const struct *inet_if_ip_info_t* *info_p)
Set the ip address, netmask and gateway of the WiFi SoftAP.

Return zero(0) or negative error code.

int **esp_wifi_softap_get_ip_info** (struct *inet_if_ip_info_t* *info_p)
Get the SoftAP ip address, netmask and gateway.

Return zero(0) or negative error code.

Parameters

- info_p: Read ip information.

int **esp_wifi_softap_get_number_of_connected_stations** (void)
Get the number of stations connected to the SoftAP.

Return Number of conencted stations.

int **esp_wifi_softap_get_station_info** (struct *esp_wifi_softap_station_info_t* *info_p, int length)
Get the information of stations connected to the SoftAP, including MAC and IP addresses.

Return Number of valid station information entries or negative error code.

Parameters

- info_p: An array to write the station information to.
- length: Length of the info array.

int **esp_wifi_softap_dhcp_server_start** (void)
Enable the SoftAP DHCP server.

Return zero(0) or negative error code.

int **esp_wifi_softap_dhcp_server_stop** (void)
Disable the SoftAP DHCP server. The DHCP server is enabled by default.

Return zero(0) or negative error code.

enum *esp_wifi_dhcp_status_t* **esp_wifi_softap_dhcp_server_status** (void)
Get the SoftAP DHCP server status.

Return DHCP server status.

struct **esp_wifi_softap_station_info_t**
#include <softap.h>

Public Members

uint8_t **bssid**[6]

struct *inet_ip_addr_t* **ip_address**

esp_wifi_station — Espressif WiFi Station

This module is a wrapper for the Espressif WiFi station interface.

Source code: `src/drivers/esp_wifi/station.h`, `src/drivers/esp_wifi/station.c`

Test code: `tst/drivers/esp_wifi/station/main.c`

Enums

`enum esp_wifi_station_status_t`

Values:

```
esp_wifi_station_status_idle_t = 0
esp_wifi_station_status_connecting_t
esp_wifi_station_status_wrong_password_t
esp_wifi_station_status_no_ap_found_t
esp_wifi_station_status_connect_fail_t
esp_wifi_station_status_got_ip_t
```

Functions

`int esp_wifi_station_init (const char *ssid_p, const char *password_p, const struct inet_if_ip_info_t *info_p)`

Initialize the WiFi station.

Return zero(0) or negative error code.

Parameters

- `ssid_p`: WiFi SSID to connect to.
- `password_p`: WiFi password.
- `info_p`: Static ip configuration or NULL to use DHCP.

`int esp_wifi_station_connect (void)`

Connect the WiFi station to the Access Point (AP).

Return zero(0) or negative error code.

`int esp_wifi_station_disconnect (void)`

Disconnect the WiFi station from the AP.

Return zero(0) or negative error code.

`int esp_wifi_station_set_ip_info (const struct inet_if_ip_info_t *info_p)`

Set the ip address, netmask and gateway of the WiFi station.

Return zero(0) or negative error code.

int **esp_wifi_station_get_ip_info** (struct *inet_if_ip_info_t* *info_p)
Get the station ip address, netmask and gateway.

Return zero(0) or negative error code.

int **esp_wifi_station_set_reconnect_policy** (int policy)
Set whether the station will reconnect to the AP after disconnection. It will do so by default.

Return zero(0) or negative error code.

Parameters

- policy: If it's true, it will enable reconnection; if it's false, it will disable reconnection.

int **esp_wifi_station_get_reconnect_policy** (void)
Check whether the station will reconnect to the AP after disconnection.

Return true(1) or false(0).

enum *esp_wifi_station_status_t* **esp_wifi_station_get_connect_status** (void)
Get the connection status of the WiFi station.

Return The connection status.

int **esp_wifi_station_dhcp_client_start** (void)
Enable the station DHCP client.

Return zero(0) or negative error code.

int **esp_wifi_station_dhcp_client_stop** (void)
Disable the station DHCP client.

Return zero(0) or negative error code.

enum *esp_wifi_dhcp_status_t* **esp_wifi_station_dhcp_client_status** (void)
Get the station DHCP client status.

Return Station DHCP client status.

Source code: [src/drivers/esp_wifi.h](#), [src/drivers/esp_wifi.c](#)

Test code: [tst/drivers/esp_wifi/main.c](#)

Enums

enum `esp_wifi_op_mode_t`

Values:

`esp_wifi_op_mode_null_t` = 0

`esp_wifi_op_mode_station_t`

`esp_wifi_op_mode_softap_t`

`esp_wifi_op_mode_station_softap_t`

`esp_wifi_op_mode_max_t`

enum `esp_wifi_phy_mode_t`

Physical WiFi mode.

Values:

`esp_wifi_phy_mode_11b_t` = 1

`esp_wifi_phy_mode_11g_t`

`esp_wifi_phy_mode_11n_t`

enum `esp_wifi_dhcp_status_t`

DHCP status.

Values:

`esp_wifi_dhcp_status_stopped_t` = 0

`esp_wifi_dhcp_status_running_t`

Functions

int `esp_wifi_module_init` (void)

Initialize the Espressif WiFi module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

int `esp_wifi_set_op_mode` (enum *esp_wifi_op_mode_t* mode)

Set the WiFi operating mode to None, Station, SoftAP or Station + SoftAP. The default mode is SoftAP.

Return zero(0) or negative error code.

Parameters

- mode: Operating mode to set.

enum *esp_wifi_op_mode_t* `esp_wifi_get_op_mode` (void)

Get the current WiFi operating mode. The operating mode can be None, Station, SoftAP, or Station + SoftAP.

Return Current operating mode.

int **esp_wifi_set_phy_mode** (enum *esp_wifi_phy_mode_t* mode)
Set the WiFi physical mode (802.11b/g/n).

The SoftAP only supports b/g.

Return zero(0) or negative error code.

Parameters

- mode: Physical mode.

enum *esp_wifi_phy_mode_t* **esp_wifi_get_phy_mode** (void)
Get the physical mode (802.11b/g/n).

Return WiFi physical mode.

void **esp_wifi_print** (void *chout_p)
Print information about the WiFi.

exti — External interrupts

Source code: [src/drivers/exti.h](#), [src/drivers/exti.c](#)

Test code: [tst/drivers/exti/main.c](#)

Defines

EXTI_TRIGGER_BOTH_EDGES
Trigger an interrupt on both rising and falling edges.

EXTI_TRIGGER_FALLING_EDGE
Trigger an interrupt on falling edges.

EXTI_TRIGGER_RISING_EDGE
Trigger an interrupt on both rising edges.

Functions

int **exti_module_init** (void)
Initialize the external interrupt (EXTI) module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

int **exti_init** (struct exti_driver_t *self_p, struct exti_device_t *dev_p, int trigger, void (*on_interrupt)) void *arg_p, void *arg_p Initialize given driver object.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object to be initialized.
- `dev_p`: Device to use.
- `trigger`: One of `EXTI_TRIGGER_BOTH_EDGES`, `EXTI_TRIGGER_FALLING_EDGE` or `EXTI_TRIGGER_RISING_EDGE`.
- `on_interrupt`: Function callback called when an interrupt occurs.
- `arg_p`: Function callback argument.

int **exti_start** (**struct** exti_driver_t **self_p*)
Starts the EXTI device using given driver object.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object.

int **exti_stop** (**struct** exti_driver_t **self_p*)
Stops the EXTI device referenced by given driver object.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object.

int **exti_clear** (**struct** exti_driver_t **self_p*)
Clear the interrupt flag.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object.

Variables

struct exti_device_t **exti_device**[EXTI_DEVICE_MAX]

flash — Flash memory

Source code: [src/drivers/flash.h](#), [src/drivers/flash.c](#)

Test code: [tst/drivers/flash/main.c](#)

Functions

int **flash_module_init** (void)
Initialize the flash module. This function must be called before calling any other function in this module.
The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

int **flash_init** (struct flash_driver_t *self_p, struct flash_device_t *dev_p)
Initialize given driver object.

Return zero(0) or negative error code.

Parameters

- self_p: Driver object to initialize.
- dev_p: Device to use.

ssize_t **flash_read** (struct flash_driver_t *self_p, void *dst_p, uintptr_t src, size_t size)
Read data from given flash memory.

Return Number of read bytes or negative error code.

Parameters

- self_p: Initialized driver object.
- dst_p: Buffer to read into.
- src: Address in flash memory to read from.
- size: Number of bytes to receive.

ssize_t **flash_write** (struct flash_driver_t *self_p, uintptr_t dst, const void *src_p, size_t size)
Write data to given flash memory. Only erased parts of the memory can be written to.

Return Number of written bytes or negative error code.

Parameters

- self_p: Initialized driver object.
- dst: Address in flash memory to write to.
- src_p: Buffer to write.
- size: Number of bytes to write.

int **flash_erase** (struct flash_driver_t *self_p, uintptr_t addr, size_t size)
Erase all sectors part of given memory range.

Return zero(0) or negative error code.

Parameters

- self_p: Initialized driver object.
- dst: Address in flash memory to erase from.
- size: Number of bytes to erase.

Variables

struct flash_device_t **flash_device**[FLASH_DEVICE_MAX]

i2c — I2C

I2C is a data transfer bus. Normally one master and one or more slaves are connected to the bus. The master addresses one slave at a time to transfer data between the devices.

The master is normally fairly easy to implement since it controls the bus clock and no race conditions can occur. The slave, on the other hand, can be implemented in various ways depending on the application requirements. In this implementation the slave will always send an acknowledgement when addressed by the master, and lock the bus by pulling SCL low until it is ready for the transmission.

Source code: [src/drivers/i2c.h](#), [src/drivers/i2c.c](#)

Test code: [tst/drivers/i2c/master/main.c](#)

Defines

I2C_BAUDRATE_3_2MBPS

I2C_BAUDRATE_1MBPS

I2C_BAUDRATE_400KBPS

I2C_BAUDRATE_100KBPS

Functions

int i2c_module_init ()

Initialize the i2c module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

int i2c_init (struct i2c_driver_t *self_p, struct i2c_device_t *dev_p, int baudrate, int address)

Initialize given driver object. The same driver object is used for both master and slave modes. Use `i2c_start ()` to start the device as a master, and `i2c_slave_start ()` to start it as a slave.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object to initialize.
- `dev_p`: I2C device to use.
- `baudrates`: Bus baudrate when in master mode. Unused in slave mode.
- `address`: Slave address when in slave mode. Unused in master mode.

int i2c_start (struct i2c_driver_t *self_p)

Start given driver object in master mode. Enables data reception and transmission, but does not start any transmission. Use `i2c_read ()` and `i2c_write ()` to exchange data with the peer.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object to initialize.

int **i2c_stop** (struct i2c_driver_t **self_p*)

Stop given driver object. Disables data reception and transmission in master mode.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object to initialize.

ssize_t **i2c_read** (struct i2c_driver_t **self_p*, int *address*, void **buf_p*, size_t *size*)

Read given number of bytes into given buffer from given slave.

Return Number of bytes read or negative error code.

Parameters

- `self_p`: Driver object.
- `address`: Slave address to read from.
- `buf_p`: Buffer to read into.
- `size`: Number of bytes to read.

ssize_t **i2c_write** (struct i2c_driver_t **self_p*, int *address*, const void **buf_p*, size_t *size*)

Write given number of bytes from given buffer to given slave.

Return Number of bytes written or negative error code.

Parameters

- `self_p`: Driver object.
- `address`: Slave address to write to.
- `buf_p`: Buffer to write.
- `size`: Number of bytes to write.

int **i2c_scan** (struct i2c_driver_t **self_p*, int *address*)

Scan the i2c bus for a slave with given address.

Return true(1) if a slave responded to given address, otherwise false(0) or negative error code.

Parameters

- `self_p`: Driver object.
- `address`: Address of the slave to scan for.

int **i2c_slave_start** (struct i2c_driver_t **self_p*)

Start given driver object in slave mode. Enables data reception and transmission, but does not start any transmission. Data transfers are started by calling the `i2c_slave_read()` and `i2c_slave_write()`.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object to initialize.

int **i2c_slave_stop** (struct i2c_driver_t *self_p)

Stop given driver object. Disables data reception and transmission in slave mode.

Return zero(0) or negative error code.

Parameters

- self_p: Driver object to initialize.

ssize_t **i2c_slave_read** (struct i2c_driver_t *self_p, void *buf_p, size_t size)

Read into given buffer from the next master that addresses this slave.

Return Number of bytes read or negative error code.

Parameters

- self_p: Driver object.
- buf_p: Buffer to read into.
- size: Number of bytes to read.

ssize_t **i2c_slave_write** (struct i2c_driver_t *self_p, const void *buf_p, size_t size)

Write given buffer to the next master that addresses this slave.

Return Number of bytes written or negative error code.

Parameters

- self_p: Driver object.
- buf_p: Buffer to write.
- size: Number of bytes to write.

Variables

struct i2c_device_t **i2c_device**[I2C_DEVICE_MAX]

i2c_soft — Software I2C

Source code: [src/drivers/i2c_soft.h](#), [src/drivers/i2c_soft.c](#)

Test code: [tst/drivers/i2c/master_soft/main.c](#)

Functions

int **i2c_soft_module_init** (void)

Initialize the i2c soft module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

```
int i2c_soft_init (struct i2c_soft_driver_t *self_p, struct pin_device_t *scl_dev_p, struct
                  pin_device_t *sda_dev_p, long baudrate, long max_clock_stretching_us, long
                  clock_stretching_sleep_us)
```

Initialize given driver object.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object to initialize.
- `scl_dev_p`: The I2C clock pin (SCL).
- `sda_dev_p`: The I2C data pin (SDA).
- `baudrate`: Bus baudrate.
- `max_clock_stretching_us`: Maximum number of microseconds to wait for the clock stretching to end.
- `clock_stretching_sleep_us`: SCL poll interval in number of microseconds waiting for clock stretching to end.

```
int i2c_soft_start (struct i2c_soft_driver_t *self_p)
```

Start given driver object. Enables data reception and transmission, but does not start any transmission. Data transfers are started by calling the `i2c_soft_read()` and `i2c_soft_write()`.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object to initialize.

```
int i2c_soft_stop (struct i2c_soft_driver_t *self_p)
```

Stop given driver object. Disables data reception and transmission.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object to initialize.

```
ssize_t i2c_soft_read (struct i2c_soft_driver_t *self_p, int address, void *buf_p, size_t size)
```

Read given number of bytes into given buffer from given slave.

Return Number of bytes read or negative error code.

Parameters

- `self_p`: Driver object.
- `address`: Slave address to read from.
- `buf_p`: Buffer to read into.
- `size`: Number of bytes to read.

```
ssize_t i2c_soft_write (struct i2c_soft_driver_t *self_p, int address, const void *buf_p, size_t size)
```

Write given number of bytes from given buffer to given slave.

Return Number of bytes written or negative error code.

Parameters

- `self_p`: Driver object.
- `address`: Slave address to write to.
- `buf_p`: Buffer to write.
- `size`: Number of bytes to write.

int **i2c_soft_scan** (struct *i2c_soft_driver_t* **self_p*, int *address*)

Scan the i2c bus for a slave with given address.

Return true(1) if a slave responded to given address, otherwise false(0) or negative error code.

Parameters

- `self_p`: Driver object.
- `address`: Address of the slave to scan for.

struct **i2c_soft_driver_t**

#include <i2c_soft.h>

Public Members

struct pin_device_t ***scl_p**

struct pin_device_t ***sda_p**

long **baudrate**

long **baudrate_us**

long **max_clock_stretching_us**

long **clock_stretching_sleep_us**

mcp2515 — CAN BUS chipset

Source code: [src/drivers/mcp2515.h](#), [src/drivers/mcp2515.c](#)

Test code: [tst/drivers/mcp2515/main.c](#)

Defines

MCP2515_SPEED_1000KBPS

MCP2515_SPEED_500KBPS

MCP2515_MODE_NORMAL

MCP2515_MODE_LOOPBACK

Functions

int **mcp2515_init** (**struct** *mcp2515_driver_t* **self_p*, **struct** *spi_device_t* **spi_p*, **struct** *pin_device_t* **cs_p*, **struct** *exti_device_t* **exti_p*, void **chin_p*, int *mode*, int *speed*)
Initialize given driver object.

Return zero(0) or negative error code.

Parameters

- *self_p*: Driver object to initialize.
- *spi_p*: SPI driver to use.
- *cs_p*: SPI chip select pin.
- *exti_p*: External interrupt to use.
- *chin_p*: Frames received from the hardware are written to this channel.
- *mode*: Device mode.
- *speed*: CAN bus speed in kbps.

int **mcp2515_start** (**struct** *mcp2515_driver_t* **self_p*)
Starts the CAN device using given driver object.

Return zero(0) or negative error code.

Parameters

- *self_p*: Initialized driver object.

int **mcp2515_stop** (**struct** *mcp2515_driver_t* **self_p*)
Stops the CAN device referenced by driver object.

Return zero(0) or negative error code.

Parameters

- *self_p*: Initialized driver object.

ssize_t **mcp2515_read** (**struct** *mcp2515_driver_t* **self_p*, **struct** *mcp2515_frame_t* **frame_p*)
Read a CAN frame.

Return zero(0) or negative error code.

Parameters

- *self_p*: Initialized driver object.
- *frame_p*: Read frame.

ssize_t **mcp2515_write** (**struct** *mcp2515_driver_t* **self_p*, **const struct** *mcp2515_frame_t* **frame_p*)
Write a CAN frame.

Return zero(0) or negative error code.

Parameters

- *self_p*: Initialized driver object.
- *frame_p*: Frame to write.

```
struct mcp2515_frame_t
```

Public Members

```
uint32_t id
int size
int rtr
uint32_t timestamp
uint8_t data[8]
```

```
struct mcp2515_driver_t
```

Public Functions

```
mcp2515_driver_t::THRSTACK(stack, 1024)
```

Public Members

```
struct spi_driver_t spi
struct exti_driver_t exti
int mode
int speed
struct chan_t chout
struct chan_t *chin_p
struct sem_t isr_sem
struct sem_t tx_sem
```

nrf24l01 — Wireless communication

Source code: [src/drivers/nrf24l01.h](#), [src/drivers/nrf24l01.c](#)

Functions

```
int nrf24l01_module_init (void)
```

Initialize NRF24L01 module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

```
int nrf24l01_init (struct nrf24l01_driver_t *self_p, struct spi_device_t *spi_p, struct pin_device_t
                  *cs_p, struct pin_device_t *ce_p, struct exti_device_t *exti_p, uint32_t address)
```

Initialize given driver object from given configuration.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object to be initialized.
- `spi_p`: SPI device.
- `cs_p`: Chip select pin device.
- `ce_p`: CE pin device.
- `exti_p`: External interrupt flag device.
- `address`: 4 MSB:s of RX pipes. LSB is set to 0 through 5 for the 6 pipes.

int **nrf24l01_start** (struct *nrf24l01_driver_t* **self_p*)
Starts the NRF24L01 device using given driver object.

Return zero(0) or negative error code.

Parameters

- `self_p`: Initialized driver object.

int **nrf24l01_stop** (struct *nrf24l01_driver_t* **self_p*)
Stops the NRF24L01 device referenced by driver object.

Return zero(0) or negative error code.

Parameters

- `self_p`: Initialized driver object.

ssize_t **nrf24l01_read** (struct *nrf24l01_driver_t* **self_p*, void **buf_p*, size_t *size*)
Read data from the NRF24L01 device.

Return Number of received bytes or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `buf_p`: Buffer to read into.
- `size`: Number of bytes to read (must be 32).

ssize_t **nrf24l01_write** (struct *nrf24l01_driver_t* **self_p*, uint32_t *address*, uint8_t *pipe*, const void **buf_p*, size_t *size*)
Write data to the NRF24L01 device.

Return number of sent bytes or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `address`: 4 MSB:s of TX address.
- `pipe`: LSB of TX address.
- `buf_p`: Buffer to write.
- `size`: Number of bytes to write (must be 32).

```
struct nrf24l01_driver_t
    #include <nrf24l01.h>
```

Public Members

```
struct spi_driver_t spi
struct exti_driver_t exti
struct pin_driver_t ce
struct queue_t irqchan
struct queue_t chin
struct thrd_t *thrd_p
uint32_t address
char irqbuf[8]
char chinbuf[32]
char stack[256]
```

owi — One-Wire Interface

Source code: [src/drivers/owi.h](#), [src/drivers/owi.c](#)

Test code: [tst/drivers/owi/main.c](#)

Defines

```
OWI_SEARCH_ROM
OWI_READ_ROM
OWI_MATCH_ROM
OWI_SKIP_ROM
OWI_ALARM_SEARCH
```

Functions

```
int owi_init (struct owi_driver_t *self_p, struct pin_device_t *dev_p, struct owi_device_t *devices_p,
              size_t nmemb)
    Initialize driver object.
```

Return zero(0) or negative error code.

Parameters

- self_p: Driver object to be initialized.
- dev_p: Pin device to use.
- devices_p: Storage for devices found when searching.

- `nmemb`: Number of members in devices.

int **owi_reset** (struct *owi_driver_t* **self_p*)

Send reset on one wire bus.

Return true(1) if one or more devices are connected to the bus, false(0) if no devices were found, otherwise negative error code.

Parameters

- `self_p`: Driver object.

int **owi_search** (struct *owi_driver_t* **self_p*)

Search for devices on given one wire bus. The device id of all found devices are stored in the devices array passed to `owi_init()`.

Return Number of devices found or negative error code.

Parameters

- `self_p`: Driver object.

ssize_t **owi_read** (struct *owi_driver_t* **self_p*, void **buf_p*, size_t *size*)

Read into buffer from one wire bus.

Return Number of bits read or negative error code.

Parameters

- `self_p`: Driver object.
- `buf_p`: Buffer to read into.
- `size`: Number of bits to read.

ssize_t **owi_write** (struct *owi_driver_t* **self_p*, const void **buf_p*, size_t *size*)

Write buffer to given one wire bus.

Return Number of bits written or negative error code.

Parameters

- `self_p`: Driver object.
- `buf_p`: Buffer to write.
- `size`: Number of bits to write.

struct **owi_device_t**

Public Members

uint8_t `id`[8]

struct **owi_driver_t**

Public Members

```
struct pin_driver_t pin
struct owi_device_t *devices_p
size_t nmemb
size_t len
```

pin — Digital pins

Debug file system commands

Three debug file system commands are available, all located in the directory `drivers/pin/`. These commands directly access the pin device registers, without using the pin driver object.

Command	Description
<code>set_mode <pin> <mode></code>	Set the mode of the pin <pin> to <mode>, where <mode> is one of output and input.
<code>read <pin></code>	Read current input or output value of the pin <pin>. high or low is printed.
<code>write <pin> <value></code>	Write the value <value> to pin <pin>, where <value> is one of high and low.

Example output from the shell:

```
$ drivers/pin/set_mode d2 output
$ drivers/pin/read d2
low
$ drivers/pin/write d2 high
$ drivers/pin/read d2
high
$ drivers/pin/set_mode d3 input
$ drivers/pin/read d3
low
```

Source code: [src/drivers/pin.h](#), [src/drivers/pin.c](#)

Test code: [tst/drivers/pin/main.c](#)

Defines

PIN_OUTPUT

PIN_INPUT

Configure the pin as an input pin.

Functions

```
int pin_module_init (void)
```

Initialize the pin module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

int **pin_init** (struct pin_driver_t *self_p, struct pin_device_t *dev_p, int mode)
Initialize given driver object with given device and mode.

Return zero(0) or negative error code.

Parameters

- self_p: Driver object to be initialized.
- dev_p: Device to use.
- mode: Pin mode. One of PIN_INPUT or PIN_OUTPUT.

int **pin_write** (struct pin_driver_t *self_p, int value)
Write given value to given pin.

Return zero(0) or negative error code.

Parameters

- self_p: Driver object.
- value: 1 for high and 0 for low output.

int **pin_read** (struct pin_driver_t *self_p)
Read the current value of given pin.

Return 1 for high and 0 for low input, otherwise negative error code.

Parameters

- self_p: Driver object.

int **pin_toggle** (struct pin_driver_t *self_p)
Toggle the pin output value (high/low).

Return zero(0) or negative error code.

Parameters

- self_p: Driver object.

int **pin_set_mode** (struct pin_driver_t *self_p, int mode)
Set the pin mode of given pin.

Return zero(0) or negative error code.

Parameters

- self_p: Driver object.
- mode: New pin mode.

static int **pin_device_set_mode** (const struct pin_device_t *dev_p, int mode)
Pin device mode to set. One of PIN_INPUT or PIN_OUTPUT.

Return zero(0) or negative error code.

Parameters

- `self_p`: Pin device.
- `mode`: New pin mode.

static int `pin_device_read` (`const struct pin_device_t *dev_p`)

Read the value of given pin device.

Return 1 for high and 0 for low input, otherwise negative error code.

Parameters

- `self_p`: Pin device.

static int `pin_device_write_high` (`const struct pin_device_t *dev_p`)

Write high to given pin device.

Return zero(0) or negative error code.

Parameters

- `self_p`: Pin device.

static int `pin_device_write_low` (`const struct pin_device_t *dev_p`)

Write low to given pin device.

Return zero(0) or negative error code.

Parameters

- `self_p`: Pin device.

Variables

struct pin_device_t `pin_device`[PIN_DEVICE_MAX]

pwm — Pulse width modulation

Source code: [src/drivers/pwm.h](#), [src/drivers/pwm.c](#)

Functions

int `pwm_init` (`struct pwm_driver_t *self_p`, `struct pwm_device_t *dev_p`)

Initialize driver object.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object to be initialized.
- `dev_p`: Device to use.

int `pwm_set_duty` (`struct pwm_driver_t *self_p`, `uint8_t value`)

Set the duty cycle.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object.
- `value`: Value to set [0..255].

int **pwm_get_duty** (struct pwm_driver_t **self_p*)
Get current duty cycle.

Return Value in the range [0..255], or negative error code.

Parameters

- `self_p`: Driver object.

struct pwm_device_t ***pwm_pin_to_device** (struct pin_device_t **pin_p*)
Get the PWM device for given pin.

Return PWM device, or NULL on error.

Parameters

- `pin_p`: The pin device to get the pwm device for.

Variables

struct pwm_device_t **pwm_device**[PWM_DEVICE_MAX]

sd — Secure Digital memory

Source code: [src/drivers/sd.h](#), [src/drivers/sd.c](#)

Test code: [tst/drivers/sd/main.c](#)

Defines

SD_ERR_NORESPONSE_WAIT_FOR_DATA_START_BLOCK

SD_ERR_GO_IDLE_STATE

SD_ERR_CRC_ON_OFF

SD_ERR_SEND_IF_COND

SD_ERR_CHECK_PATTERN

SD_ERR_SD_SEND_OP_COND

SD_ERR_READ_OCR

SD_ERR_READ_COMMAND

SD_ERR_READ_DATA_START_BLOCK

SD_ERR_READ_WRONG_DATA_CRC

SD_ERR_WRITE_BLOCK

`SD_ERR_WRITE_BLOCK_TOKEN_DATA_RES_ACCEPTED`

`SD_ERR_WRITE_BLOCK_WAIT_NOT_BUSY`

`SD_ERR_WRITE_BLOCK_SEND_STATUS`

`SD_BLOCK_SIZE`

`SD_CCC` (`csd_p`)

`SD_C_SIZE` (`csd_p`)

`SD_C_SIZE_MULT` (`csd_p`)

`SD_SECTOR_SIZE` (`csd_p`)

`SD_WRITE_BL_LEN` (`csd_p`)

`SD_CSD_STRUCTURE_V1`

`SD_CSD_STRUCTURE_V2`

Functions

int **sd_init** (**struct** *sd_driver_t* **self_p*, **struct** spi_driver_t **spi_p*)
Initialize given driver object.

Return zero(0) or negative error code.

Parameters

- *self_p*: Driver object to initialize.

int **sd_start** (**struct** *sd_driver_t* **self_p*)
Start given SD card driver. This resets the SD card and performs the initialization sequence.

Return zero(0) or negative error code.

Parameters

- *self_p*: Initialized driver object.

int **sd_stop** (**struct** *sd_driver_t* **self_p*)
Stop given SD card driver.

Return zero(0) or negative error code.

Parameters

- *self_p*: Initialized driver object.

ssize_t **sd_read_cid** (**struct** *sd_driver_t* **self_p*, **struct** *sd_cid_t* **cid_p*)
Read card CID register. The CID contains card identification information such as Manufacturer ID, Product name, Product serial number and Manufacturing date.

Return zero(0) or negative error code.

Parameters

- *self_p*: Initialized driver object.
- *cid*: pointer to cid data store.

`ssize_t sd_read_csd (struct sd_driver_t *self_p, union sd_csd_t *csd_p)`

Read card CSD register. The CSD contains that provides information regarding access to the card's contents.

Return zero(0) or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `csd`: pointer to csd data store.

`ssize_t sd_read_block (struct sd_driver_t *self_p, void *dst_p, uint32_t src_block)`

Read given block from SD card.

Return Number of read bytes or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `buf_p`: Buffer to read into.
- `src_block`: Block to read from.

`ssize_t sd_write_block (struct sd_driver_t *self_p, uint32_t dst_block, const void *src_p)`

Write data to the SD card.

Return Number of written bytes or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `dst_block`: Block to write to.
- `src_p`: Buffer to write.

Variables

`struct sd_csd_v2_t PACKED`

`struct sd_cid_t`

Public Members

`uint8_t mid`

`char oid[2]`

`char pnm[5]`

`uint8_t prv`

`uint32_t psn`

`uint16_t mdt`

`uint8_t crc`

`struct sd_csd_v1_t`

Public Members

`uint8_t reserved1`
`uint8_t csd_structure`
`uint8_t taac`
`uint8_t nsac`
`uint8_t tran_speed`
`uint8_t ccc_high`
`uint8_t read_bl_len`
`uint8_t ccc_low`
`uint8_t c_size_high`
`uint8_t reserved2`
`uint8_t dsr_imp`
`uint8_t read_blk_misalign`
`uint8_t write_blk_misalign`
`uint8_t read_bl_partial`
`uint8_t c_size_mid`
`uint8_t vdd_r_curr_max`
`uint8_t vdd_r_curr_min`
`uint8_t c_size_low`
`uint8_t c_size_mult_high`
`uint8_t vdd_w_curr_max`
`uint8_t vdd_w_curr_min`
`uint8_t sector_size_high`
`uint8_t erase_blk_en`
`uint8_t c_size_mult_low`
`uint8_t wp_grp_size`
`uint8_t sector_size_low`
`uint8_t write_bl_len_high`
`uint8_t r2w_factor`
`uint8_t reserved3`
`uint8_t wp_grp_enable`
`uint8_t reserved4`
`uint8_t write_bl_partial`
`uint8_t write_bl_len_low`
`uint8_t reserved5`
`uint8_t file_format`

```
uint8_t tmp_write_protect
uint8_t perm_write_protect
uint8_t copy
uint8_t file_format_grp
uint8_t crc
struct sd_csd_v2_t
```

Public Members

```
uint8_t reserved1
uint8_t csd_structure
uint8_t taac
uint8_t nsac
uint8_t tran_speed
uint8_t ccc_high
uint8_t read_bl_len
uint8_t ccc_low
uint8_t reserved2
uint8_t dsr_imp
uint8_t read_blk_misalign
uint8_t write_blk_misalign
uint8_t read_bl_partial
uint8_t c_size_high
uint8_t reserved3
uint8_t c_size_mid
uint8_t c_size_low
uint8_t sector_size_high
uint8_t erase_blk_en
uint8_t reserved4
uint8_t wp_grp_size
uint8_t sector_size_low
uint8_t write_bl_len_high
uint8_t r2w_factor
uint8_t reserved5
uint8_t wp_grp_enable
uint8_t reserved6
uint8_t write_bl_partial
```

```
uint8_t write_bl_len_low
uint8_t reserved7
uint8_t file_format
uint8_t tmp_write_protect
uint8_t perm_write_protect
uint8_t copy
uint8_t file_format_grp
uint8_t crc
```

```
union sd_csd_t
```

Public Members

```
struct sd_csd_v1_t v1
struct sd_csd_v2_t v2
```

```
struct sd_driver_t
```

Public Members

```
struct spi_driver_t *spi_p
int type
```

sdio — Secure Digital Input Output

Source code: [src/drivers/sdio.h](#), [src/drivers/sdio.c](#)

Defines

```
SDIO_IO_RW_EXTENDED_BLOCK_MODE_BYTE
SDIO_IO_RW_EXTENDED_BLOCK_MODE_BLOCK
SDIO_IO_RW_EXTENDED_OP_CODE_FIXED_ADDRESS
SDIO_IO_RW_EXTENDED_OP_CODE_INCREMENTING_ADDRESS
```

Functions

```
int sdio_module_init (void)
```

Initialize the SDIO module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

int **sdio_init** (struct sdio_driver_t *self_p, struct sdio_device_t *dev_p)

Initialize driver object from given configuration.

Return zero(0) or negative error code.

Parameters

- self_p: Driver object to be initialized.
- dev_p: Device to use.

int **sdio_start** (struct sdio_driver_t *self_p)

Starts the SDIO device using given driver object.

Return zero(0) or negative error code.

Parameters

- self_p: Initialized driver object.

int **sdio_stop** (struct sdio_driver_t *self_p)

Stops the SDIO device referenced by driver object.

Return zero(0) or negative error code.

Parameters

- self_p: Initialized driver object.

int **sdio_send_relative_addr** (struct sdio_driver_t *self_p)

Send the send relative address command (CMD3) to the device and optionally wait for the response.

Return zero(0) or negative error code.

Parameters

- self_p: Initialized driver object.

int **sdio_io_send_op_cond** (struct sdio_driver_t *self_p)

Send the io send operation condition command (CMD5) to the device and optionally wait for the response.

Return zero(0) or negative error code.

Parameters

- self_p: Initialized driver object.

int **sdio_select_deselect_card** (struct sdio_driver_t *self_p)

Send the select/deselect card command (CMD7) to the device and optionally wait for the response.

Return zero(0) or negative error code.

Parameters

- self_p: Initialized driver object.

int **sdio_io_read_direct** (struct sdio_driver_t *self_p, void *dst_p)

Execute the input output read write direct command (CMD52) as a read operation with given parameters.

Return Number of bytes read or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `dst_p`: Destination buffer.

int **sdio_io_write_direct** (**struct** sdio_driver_t **self_p*, **const** void **src_p*)

Execute the input output read write direct command (CMD52) as a write operation with given parameters.

Return Number of bytes written or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `src_p`: Source buffer.

ssize_t **sdio_io_read_extended** (**struct** sdio_driver_t **self_p*, int *function_number*, int *block_mode*, int *op_code*, void **dst_p*, uint32_t *src_address*, size_t *size*)

Execute the input output read write extended command (CMD53) as a read operation with given parameters.

Return Number of bytes read or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `function_number`: Function number.
- `block_mode`: Block or byte mode.
- `op_code`: Operation code.
- `dst_p`: Destination buffer.
- `src_address`: Source address.
- `size`: Number of bytes to read.

ssize_t **sdio_io_write_extended** (**struct** sdio_driver_t **self_p*, int *function_number*, int *block_mode*, int *op_code*, uint32_t *dst_address*, **const** void **src_p*, size_t *size*)

Execute the input output read write extended command (CMD53) as a write operation with given parameters.

Return Number of bytes written or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `function_number`: Function number.
- `block_mode`: Block or byte mode.
- `op_code`: Operation code.
- `dst_address`: Destination address.
- `src_p`: Source buffer.
- `size`: Number of bytes to write.

Variables

```
struct sdio_device_t sdio_device[SDIO_DEVICE_MAX]
struct sdio_io_rw_extended_t
```

Public Members

```
uint8_t rw_flag
uint8_t function_number
uint8_t block_mode
uint8_t op_code
uint8_t register_address_16_15
uint8_t register_address_14_7
uint8_t register_address_6_0
uint8_t byte_block_count_8
uint8_t byte_block_count_7_0
```

spi — Serial Peripheral Interface

Source code: [src/drivers/spi.h](#), [src/drivers/spi.c](#)

Defines

```
SPI_MODE_SLAVE
SPI_MODE_MASTER
SPI_SPEED_8MBPS
SPI_SPEED_4MBPS
SPI_SPEED_2MBPS
SPI_SPEED_1MBPS
SPI_SPEED_500KBPS
SPI_SPEED_250KBPS
SPI_SPEED_125KBPS
```

Functions

```
int spi_module_init (void)
    Initialize SPI module. This function must be called before calling any other function in this module.
    The module will only be initialized once even if this function is called multiple times.
```

Return zero(0) or negative error code.

int **spi_init** (**struct** spi_driver_t *self_p, **struct** spi_device_t *dev_p, **struct** pin_device_t *ss_pin_p, int mode, int speed, int cpol, int cpha)
Initialize driver object.

Return zero(0) or negative error code.

Parameters

- self_p: Driver object to initialize.
- dev_p: Device to use.
- ss_pin_p: Slave select pin device.
- mode: Master or slave mode.
- speed: Speed in kbps.
- cpol: Polarity, 0 or 1.
- cpha: Phase, 0 or 1.

int **spi_start** (**struct** spi_driver_t *self_p)
Start given SPI driver. Configures the SPI hardware.

Return zero(0) or negative error code.

Parameters

- self_p: Initialized driver object.

int **spi_stop** (**struct** spi_driver_t *self_p)
Stop given SPI driver. Deconfigures the SPI hardware if given driver currently owns the bus.

Return zero(0) or negative error code.

Parameters

- self_p: Initialized driver object.

int **spi_take_bus** (**struct** spi_driver_t *self_p)
In multi master application the driver must take ownership of the SPI bus before performing data transfers. Will re-configure the SPI hardware if configured by another driver.

Return zero(0) or negative error code.

Parameters

- self_p: Initialized driver object.

int **spi_give_bus** (**struct** spi_driver_t *self_p)
In multi master application the driver must give ownership of the SPI bus to let other masters take it.

Return zero(0) or negative error code.

Parameters

- self_p: Initialized driver object.

int **spi_select** (struct spi_driver_t *self_p)
 Select the slave by asserting the slave select pin.

Return zero(0) or negative error code.

Parameters

- self_p: Initialized driver object.

int **spi_deselect** (struct spi_driver_t *self_p)
 Deselect the slave by de-asserting the slave select pin.

Return zero(0) or negative error code.

Parameters

- self_p: Initialized driver object.

ssize_t **spi_transfer** (struct spi_driver_t *self_p, void *rxbuf_p, const void *txbuf_p, size_t size)
 Simultaneous read/write operation over the SPI bus.

Return Number of transferred bytes or negative error code.

Parameters

- self_p: Initialized driver object.
- rxbuf_p: Buffer to read into.
- txbuf_p: Buffer to write.
- size: Number of bytes to transfer.

ssize_t **spi_read** (struct spi_driver_t *self_p, void *buf_p, size_t size)
 Read data from the SPI bus.

Return Number of read bytes or negative error code.

Parameters

- self_p: Initialized driver object.
- buf_p: Buffer to read into.
- size: Number of bytes to receive.

ssize_t **spi_write** (struct spi_driver_t *self_p, const void *buf_p, size_t size)
 Write data to the SPI bus.

Return Number of written bytes or negative error code.

Parameters

- self_p: Initialized driver object.
- buf_p: Buffer to write.
- size: Number of bytes to write.

ssize_t **spi_get** (struct spi_driver_t *self_p, uint8_t *data_p)
 Get one byte of data from the SPI bus.

Return Number of read bytes or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `data_p`: Read data.

`ssize_t spi_put (struct spi_driver_t *self_p, uint8_t data)`
Put one byte of data to the SPI bus.

Return Number of written bytes or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `data`: data to write.

Variables

`struct spi_device_t spi_device[SPI_DEVICE_MAX]`

uart — Universal Asynchronous Receiver/Transmitter

Source code: [src/drivers/uart.h](#), [src/drivers/uart.c](#)

Test code: [tst/drivers/uart/main.c](#)

Defines

`uart_read (self_p, buf_p, size)`
Read data from the UART.

Return Number of received bytes or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `buf_p`: Buffer to read into.
- `size`: Number of bytes to receive.

`uart_write (self_p, buf_p, size)`
Write data to the UART.

Return Number of written bytes or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `buf_p`: Buffer to write.
- `size`: Number of bytes to write.

Typedefs

```
typedef int (*uart_rx_filter_cb_t) (char c)
```

Functions

```
int uart_module_init (void)
```

Initialize UART module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

```
int uart_init (struct uart_driver_t *self_p, struct uart_device_t *dev_p, long baudrate, void *rxbuf_p,  
               size_t size)
```

Initialize driver object from given configuration.

Return zero(0) or negative error code.

Parameters

- *self_p*: Driver object to be initialized.
- *dev_p*: Device to use.
- *baudrate*: Baudrate.
- *rxbuf_p*: Reception buffer.
- *size*: Reception buffer size.

```
int uart_set_rx_filter_cb (struct uart_driver_t *self_p, uart_rx_filter_cb_t rx_filter_cb)
```

Set the reception filter callback function.

Return zero(0) or negative error code.

Parameters

- *self_p*: Initialized driver object.
- *rx_filter_cb*: Callback to set.

```
int uart_start (struct uart_driver_t *self_p)
```

Starts the UART device using given driver object.

Return zero(0) or negative error code.

Parameters

- *self_p*: Initialized driver object.

```
int uart_stop (struct uart_driver_t *self_p)
```

Stops the UART device referenced by driver object.

Return zero(0) or negative error code.

Parameters

- *self_p*: Initialized driver object.

Variables

struct uart_device_t **uart_device**[UART_DEVICE_MAX]

uart_soft — Bitbang UART

Source code: [src/drivers/uart_soft.h](#), [src/drivers/uart_soft.c](#)

Defines

uart_soft_read (self_p, buf_p, size)

Read data from the UART.

Return Number of received bytes or negative error code.

Parameters

- self_p: Initialized driver object.
- buf_p: Buffer to read into.
- size: Number of bytes to receive.

uart_soft_write (self_p, buf_p, size)

Write data to the UART.

Return number of sent bytes or negative error code.

Parameters

- self_p: Initialized driver object.
- buf_p: Buffer to write.
- size: Number of bytes to write.

Functions

int **uart_soft_init** (**struct** *uart_soft_driver_t* *self_p, **struct** pin_device_t *tx_dev_p, **struct** pin_device_t *rx_dev_p, **struct** exti_device_t *rx_exti_dev_p, int baudrate, void *rxbuf_p, size_t size)

Initialize driver object from given configuration.

Return zero(0) or negative error code.

Parameters

- self_p: Driver object to be initialized.
- tx_dev_p: TX pin device.
- rx_dev_p: RX pin device.
- rx_exti_dev_p: RX pin external interrupt device.
- baudrate: Baudrate.

- `rxbuf_p`: Reception buffer.
- `size`: Reception buffer size.

```
struct uart_soft_driver_t
    #include <uart_soft.h>
```

Public Members

```
struct pin_driver_t tx_pin
struct pin_driver_t rx_pin
struct exti_driver_t rx_exti
struct chan_t chout
struct queue_t chin
int sample_time
int baudrate
```

usb — Universal Serial Bus

Source code: [src/drivers/usb.h](#), [src/drivers/usb.c](#)

Defines

```
REQUEST_TYPE_DATA_MASK
REQUEST_TYPE_DATA_DIRECTION_HOST_TO_DEVICE
REQUEST_TYPE_DATA_DIRECTION_DEVICE_TO_HOST
REQUEST_TYPE_TYPE_MASK
REQUEST_TYPE_TYPE_STANDARD
REQUEST_TYPE_TYPE_CLASS
REQUEST_TYPE_TYPE_VENDOR
REQUEST_TYPE_RECIPIENT_MASK
REQUEST_TYPE_RECIPIENT_DEVICE
REQUEST_TYPE_RECIPIENT_INTERFACE
REQUEST_TYPE_RECIPIENT_ENDPOINT
REQUEST_TYPE_RECIPIENT_OTHER
REQUEST_GET_STATUS
REQUEST_SET_ADDRESS
REQUEST_GET_DESCRIPTOR
REQUEST_SET_CONFIGURATION
```

DESCRIPTOR_TYPE_DEVICE
DESCRIPTOR_TYPE_CONFIGURATION
DESCRIPTOR_TYPE_STRING
DESCRIPTOR_TYPE_INTERFACE
DESCRIPTOR_TYPE_ENDPOINT
DESCRIPTOR_TYPE_INTERFACE_ASSOCIATION
DESCRIPTOR_TYPE_RPIPE
DESCRIPTOR_TYPE_CDC
USB_CLASS_USE_INTERFACE
USB_CLASS_AUDIO
USB_CLASS_CDC_CONTROL
USB_CLASS_HID
USB_CLASS_PHYSICAL
USB_CLASS_IMAGE
USB_CLASS_PRINTER
USB_CLASS_MASS_STORAGE
USB_CLASS_HUB
USB_CLASS_CDC_DATA
USB_CLASS_SMART_CARD
USB_CLASS_CONTENT_SECURITY
USB_CLASS_VIDEO
USB_CLASS_PERSONAL_HEALTHCARE
USB_CLASS_AUDIO_VIDEO_DEVICES
USB_CLASS_BILLBOARD_DEVICE_CLASS
USB_CLASS_DIAGNOSTIC_DEVICE
USB_CLASS_WIRELESS_CONTROLLER
USB_CLASS_MISCELLANEOUS
USB_CLASS_APPLICATION_SPECIFIC
USB_CLASS_VENDOR_SPECIFIC
ENDPOINT_ENDPOINT_ADDRESS_DIRECTION (address)
ENDPOINT_ENDPOINT_ADDRESS_NUMBER (address)
ENDPOINT_ATTRIBUTES_USAGE_TYPE (attributes)
ENDPOINT_ATTRIBUTES_SYNCHRONISATION_TYPE (attributes)
ENDPOINT_ATTRIBUTES_TRANSFER_TYPE (attributes)
ENDPOINT_ATTRIBUTES_TRANSFER_TYPE_CONTROL
ENDPOINT_ATTRIBUTES_TRANSFER_TYPE_ISOCHRONOUS

ENDPOINT_ATTRIBUTES_TRANSFER_TYPE_BULK
ENDPOINT_ATTRIBUTES_TRANSFER_TYPE_INTERRUPT
CONFIGURATION_ATTRIBUTES_BUS_POWERED
USB_CDC_LINE_CODING
USB_CDC_CONTROL_LINE_STATE
USB_CDC_SEND_BREAK
USB_MESSAGE_TYPE_ADD
USB_MESSAGE_TYPE_REMOVE

Functions

int **usb_format_descriptors** (void *out_p, uint8_t *buf_p, size_t size)
 Format the descriptors and write them to given channel.

Return zero(0) or negative error code.

Parameters

- out_p: Output channel.
- buf_p: Pointer to the descriptors to format.
- size: Number of bytes in the descriptors buffer.

struct *usb_descriptor_configuration_t* ***usb_desc_get_configuration** (uint8_t *desc_p, size_t size,
 int configuration)

Get the configuration descriptor for given configuration index.

Return Configuration or NULL on failure.

Parameters

- buf_p: Pointer to the descriptors.
- size: Number of bytes in the descriptors buffer.
- configuration: Configuration to find.

struct *usb_descriptor_interface_t* ***usb_desc_get_interface** (uint8_t *desc_p, size_t size, int configuration,
 int interface)

Get the interface descriptor for given configuration and interface index.

Return Interface or NULL on failure.

Parameters

- buf_p: Pointer to the descriptors.
- size: Number of bytes in the descriptors buffer.
- configuration: Configuration to find.
- interface: Interface to find.

```
struct usb_descriptor_endpoint_t *usb_desc_get_endpoint (uint8_t *desc_p, size_t size, int configuration, int interface, int endpoint)
```

Get the endpoint descriptor for given configuration, interface and endpoint index.

Return Endpoint or NULL on failure.

Parameters

- *buf_p*: Pointer to the descriptors.
- *size*: Number of bytes in the descriptors buffer.
- *configuration*: Configuration to find.
- *interface*: Interface to find.
- *endpoint*: Endpoint to find.

```
int usb_desc_get_class (uint8_t *buf_p, size_t size, int configuration, int interface)
```

Get the interface class.

Return

Parameters

- *buf_p*: Pointer to the descriptors.
- *size*: Number of bytes in the descriptors buffer.
- *configuration*: Configuration to find.
- *interface*: Interface to find.

Variables

```
struct usb_device_t usb_device[USB_DEVICE_MAX]
```

```
struct usb_setup_t
```

Public Members

```
uint8_t request_type
```

```
uint8_t request
```

```
uint16_t feature_selector
```

```
uint16_t zero_interface_endpoint
```

```
struct usb_setup_t::@15::@16 usb_setup_t::clear_feature
```

```
uint16_t zero0
```

```
uint16_t zero1
```

```
struct usb_setup_t::@15::@17 usb_setup_t::get_configuration
```

```
uint8_t descriptor_index
```

```
uint8_t descriptor_type
```

```
uint16_t language_id
```

```
struct usb_setup_t::@15::@18 usb_setup_t::get_descriptor
```



```

uint16_t device_address
uint16_t zero
struct usb_setup_t::@15::@19  usb_setup_t::set_address
uint16_t configuration_value
struct usb_setup_t::@15::@20  usb_setup_t::set_configuration
uint16_t value
uint16_t index
struct usb_setup_t::@15::@21  usb_setup_t::base
union usb_setup_t::@15  usb_setup_t::u
uint16_t length
struct usb_descriptor_header_t

```

Public Members

```

uint8_t length
uint8_t descriptor_type
struct usb_descriptor_device_t

```

Public Members

```

uint8_t length
uint8_t descriptor_type
uint16_t bcd_usb
uint8_t device_class
uint8_t device_subclass
uint8_t device_protocol
uint8_t max_packet_size_0
uint16_t id_vendor
uint16_t id_product
uint16_t bcd_device
uint8_t manufacturer
uint8_t product
uint8_t serial_number
uint8_t num_configurations
struct usb_descriptor_configuration_t

```

Public Members

```
uint8_t length
uint8_t descriptor_type
uint16_t total_length
uint8_t num_interfaces
uint8_t configuration_value
uint8_t configuration
uint8_t configuration_attributes
uint8_t max_power
```

```
struct usb_descriptor_interface_t
```

Public Members

```
uint8_t length
uint8_t descriptor_type
uint8_t interface_number
uint8_t alternate_setting
uint8_t num_endpoints
uint8_t interface_class
uint8_t interface_subclass
uint8_t interface_protocol
uint8_t interface
```

```
struct usb_descriptor_endpoint_t
```

Public Members

```
uint8_t length
uint8_t descriptor_type
uint8_t endpoint_address
uint8_t attributes
uint16_t max_packet_size
uint8_t interval
```

```
struct usb_descriptor_string_t
```

Public Members

```
uint8_t length
uint8_t descriptor_type
uint8_t string[256]
```

```
struct usb_descriptor_interface_association_t
```

Public Members

```
uint8_t length
uint8_t descriptor_type
uint8_t first_interface
uint8_t interface_count
uint8_t function_class
uint8_t function_subclass
uint8_t function_protocol
uint8_t function
```

```
struct usb_descriptor_cdc_header_t
```

Public Members

```
uint8_t length
uint8_t descriptor_type
uint8_t sub_type
uint16_t bcd
```

```
struct usb_descriptor_cdc_acm_t
```

Public Members

```
uint8_t length
uint8_t descriptor_type
uint8_t sub_type
uint8_t capabilities
```

```
struct usb_descriptor_cdc_union_t
```

Public Members

```
uint8_t length
uint8_t descriptor_type
```

```
uint8_t sub_type
uint8_t master_interface
uint8_t slave_interface
```

```
struct usb_descriptor_cdc_call_management_t
```

Public Members

```
uint8_t length
uint8_t descriptor_type
uint8_t sub_type
uint8_t capabilities
uint8_t data_interface
```

```
union usb_descriptor_t
```

Public Members

```
struct usb_descriptor_header_t header
struct usb_descriptor_device_t device
struct usb_descriptor_configuration_t configuration
struct usb_descriptor_interface_t interface
struct usb_descriptor_endpoint_t endpoint
struct usb_descriptor_string_t string
```

```
struct usb_cdc_line_info_t
```

Public Members

```
uint32_t dte_rate
uint8_t char_format
uint8_t parity_type
uint8_t data_bits
```

```
struct usb_message_header_t
```

Public Members

```
int type
```

```
struct usb_message_add_t
```

Public Members

```
struct usb_message_header_t header
int device
union usb_message_t
```

Public Members

```
struct usb_message_header_t header
struct usb_message_add_t add
```

usb_device — Universal Serial Bus - Device

A USB device is powered and enumerated by a USB host.

The implementation of this module aims to be simple, but yet flexible. It's possible to change the USB configuration descriptors at runtime by stopping the current driver, initialize a new driver and start the new driver. For simple devices only a single configuration is normally needed.

Using the USB device module is fairly easy. First write the USB descriptors, then initialize the class drivers, then initialize the USB device driver and then start it.

See the test code below for an example usage.

Class driver modules:

usb_device_class_cdc — CDC ACM (serial port over USB)

USB CDC (Communications Device Class) ACM (Abstract Control Model) is a vendor-independent publicly documented protocol that can be used for emulating serial ports over USB.

More information on [Wikipedia](#).

Source code: [src/drivers/usb/device/class/cdc.h](#), [src/drivers/usb/device/class/cdc.c](#)

Test code: [tst/drivers/usb_device/main.c](#)

Defines

usb_device_class_cdc_read (self_p, buf_p, size)
Read data from the CDC driver.

Return Number of bytes read or negative error code.

Parameters

- *self_p*: Initialized driver object.

- `buf_p`: Buffer to read into.
- `size`: Number of bytes to read.

usb_device_class_cdc_write (self_p, buf_p, size)
Write data to the CDC driver.

Return Number of bytes written or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `buf_p`: Buffer to write.
- `size`: Number of bytes to write.

Functions

```
int usb_device_class_cdc_module_init (void)
```

Initialize the CDC module.

Return zero(0) or negative error code.

```
int usb_device_class_cdc_init (struct usb_device_class_cdc_driver_1 *self_p, int control_interface,
                              int endpoint_in, int endpoint_out, void *rxbuf_p, size_t size)
    Initialize driver object from given configuration.
```

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object to be initialized.
- `rxbuf_p`: Reception buffer.
- `size`: Reception buffer size.

int usb_device_class_cdc_input_isr (struct *usb_device_class_cdc_driver_t* **self_p*)

Called by the USB device driver periodically to let the CDC driver read received data from the hardware.

Return zero(0) or negative error code.

Parameters

- `self_p`: Initialized driver object.

```
int usb_device_class_cdc_is_connected(struct usb_device_class_cdc_driver_1 *self_p)
    Check if the CDC is connected to the remote endpoint.
```

Return true(1) if connected, false(0) if disconnected, otherwise negative error code.

Parameters

- `self_p`: Initialized driver object.

```
struct usb_device_class_cdc_driver_t
#include <cdc.h>
```

Public Members

```

struct usb_device_driver_base_t base
struct usb_device_driver_t *drv_p
int control_interface
int endpoint_in
int endpoint_out
int line_state
struct usb_cdc_line_info_t line_info
struct chan_t chout
struct queue_t chin

```

Source code: `src/drivers/usb_device.h`, `src/drivers/usb_device.c`

Test code: `tst/drivers/usb_device/main.c`

Functions

int **usb_device_module_init** (void)

int **usb_device_init**(**struct** **usb_device_driver_t** * **self_p**, **struct** **usb_device_t** * **dev_p**, s
 Initialize the USB device driver object from given configuration.

Return zero(0) or negative error code.

Parameters

- **self_p**: Driver object to be initialized.
- **dev_p**: USB device to use.
- **drivers_pp**: An array of initialized drivers.
- **drivers_max**: Length of the drivers array.
- **descriptors_pp**: A NULL terminated array of USB descriptors.

int **usb_device_start** (**struct** **usb_device_driver_t** ***self_p**)
 Start the USB device device using given driver object.

Return zero(0) or negative error code.

Parameters

- **self_p**: Initialized driver object.

int **usb_device_stop** (**struct** **usb_device_driver_t** ***self_p**)
 Stop the USB device device referenced by driver object.

Return zero(0) or negative error code.

Parameters

- `self_p`: Initialized driver object.

`ssize_t usb_device_write(struct usb_device_driver_t *self_p, int endpoint, const void *buf_p, size_t size)`
Write data to given endpoint.

Return Number of bytes written or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `endpoint`: Endpoint to write to.
- `buf_p`: Buffer to write.
- `size`: Number of bytes to write.

`ssize_t usb_device_read_isr(struct usb_device_driver_t *self_p, int endpoint, void *buf_p, size_t size)`
Read data from given endpoint from an isr or with the system lock taken.

Return Number of bytes read or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `endpoint`: Endpoint to read data from.
- `buf_p`: Buffer to read into.
- `size`: Number of bytes to read.

`ssize_t usb_device_write_isr(struct usb_device_driver_t *self_p, int endpoint, const void *buf_p, size_t size)`
Write data to given endpoint from an isr or with the system lock taken.

Return Number of bytes written or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `endpoint`: Endpoint to write to.
- `buf_p`: Buffer to write.
- `size`: Number of bytes to write.

usb_host — Universal Serial Bus - Host

A USB host powers the bus and enumerates connected USB devices.

Class driver modules:

usb_host_class_hid — Human Interface Device (HID)

In computing, the USB human interface device class (USB HID class) is a part of the USB specification for computer peripherals: it specifies a device class (a type of computer hardware) for human interface devices such as keyboards, mice, game controllers and alphanumeric display devices.

More information on [Wikipedia](#).

Source code: [src/drivers/usb/host/class/hid.h](#), [src/drivers/usb/host/class/hid.c](#)

Defines

USB_CLASS_HID_SUBCLASS_NONE
USB_CLASS_HID_SUBCLASS_BOOT_INTERFACE
USB_CLASS_HID_PROTOCOL_NONE
USB_CLASS_HID_PROTOCOL_KEYBOARD
USB_CLASS_HID_PROTOCOL_MOUSE

Functions

int **usb_host_class_hid_init** (**struct** *usb_host_class_hid_driver_t* *self_p, **struct** usb_host_driver_t *usb_p, **struct** *usb_host_class_hid_device_t* *devices_p, size_t length)
Initialize driver object from given configuration.

Return zero(0) or negative error code.

Parameters

- self_p: Driver object to be initialized.
- usb_p: USB driver to use.
- devices_p: Array of devices. One entry in this array is allocated for each HID device that is connected to the host.
- length: Length of the devices array.

int **usb_host_class_hid_start** (**struct** *usb_host_class_hid_driver_t* *self_p)
Starts the HID driver.

Return zero(0) or negative error code.

Parameters

- self_p: Initialized driver object to start.

int **usb_host_class_hid_stop** (**struct** *usb_host_class_hid_driver_t* *self_p)
Stops the HID driver.

Return zero(0) or negative error code.

Parameters

- `self_p`: Initialized to stop.

```
struct usb_host_class_hid_device_t
```

Public Members

```
uint8_t buf[1]
```

```
struct usb_host_class_hid_driver_t
```

Public Members

```
struct usb_host_driver_t *usb_p
```

```
struct usb_host_class_hid_device_t *devices_p
```

```
size_t length
```

```
size_t size
```

```
struct usb_host_class_hid_driver_t::@13  usb_host_class_hid_driver_t::report
```

```
struct usb_host_device_driver_t device_driver
```

usb_host_class_mass_storage — Mass Storage

The USB mass storage device class (also known as USB MSC or UMS) is a set of computing communications protocols defined by the USB Implementers Forum that makes a USB device accessible to a host computing device and enables file transfers between the host and the USB device. To a host, the USB device acts as an external hard drive; the protocol set interfaces with a number of storage devices.

More information on [Wikipedia](#).

Source code: [src/drivers/usb/host/class/mass_storage.h](#), [src/drivers/usb/host/class/mass_storage.c](#)

Functions

```
int usb_host_class_mass_storage_init (struct usb_host_class_mass_storage_driver_t
                                     *self_p, struct usb_host_driver_t *usb_p, struct
                                     usb_host_class_mass_storage_device_t *devices_p,
                                     size_t length)
```

```
int usb_host_class_mass_storage_start (struct usb_host_class_mass_storage_driver_t *self_p)
```

```
int usb_host_class_mass_storage_stop (struct usb_host_class_mass_storage_driver_t *self_p)
```

```
ssize_t usb_host_class_mass_storage_device_read (struct usb_host_device_t *device_p, void
                                                  *buf_p, size_t address, size_t size)
```

```
struct usb_host_class_mass_storage_device_t
#include <mass_storage.h>
```

Public Members

`uint8_t buf[1]`

`struct usb_host_class_mass_storage_driver_t`

Public Members

`struct usb_host_driver_t *usb_p`

`struct usb_host_class_mass_storage_device_t *devices_p`

`size_t length`

`size_t size`

`struct usb_host_class_mass_storage_driver_t::@14 usb_host_class_mass_storage_driver_t`

`struct usb_host_device_driver_t device_driver`

Source code: [src/drivers/usb_host.h](#), [src/drivers/usb_host.c](#)

Defines

`USB_HOST_DEVICE_STATE_NONE`

`USB_HOST_DEVICE_STATE_ATTACHED`

`USB_PIPE_TYPE_CONTROL`

`USB_PIPE_TYPE_INTERRUPT`

`USB_PIPE_TYPE_ISOCHRONOUS`

`USB_PIPE_TYPE_BULK`

Functions

`int usb_host_module_init (void)`

Initialize the USB host module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

`int usb_host_init (struct usb_host_driver_t *self_p, struct usb_device_t *dev_p, struct usb_host_device_t *devices_p, size_t length)`

Initialize the USB host driver object from given configuration.

Return zero(0) or negative error code.

Parameters

- `self_p`: Driver object to be initialized.

- `dev_p`: USB device to use.
- `devices_p`: An array of devices. One entry in this array is allocated for each USB device that is connected to the host.
- `length`: Length of the devices array.

int **usb_host_start** (**struct** usb_host_driver_t **self_p*)
Start the USB host device using given driver object.

Return zero(0) or negative error code.

Parameters

- `self_p`: Initialized driver object.

int **usb_host_stop** (**struct** usb_host_driver_t **self_p*)
Stop the USB host device referenced by driver object.

Return zero(0) or negative error code.

Parameters

- `self_p`: Initialized driver object.

int **usb_host_driver_add** (**struct** usb_host_driver_t **self_p*, **struct** *usb_host_device_driver_t* **driver_p*,
void **arg_p*)
Add given class/vendor driver to the USB host driver.

When a USB device is plugged in, its class and vendor information is read by the host. Those values are used to find the device driver for this particular device. If there is no driver, the device cannot be configured and will not work.

Return zero(0) or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `driver_p`: USB device driver to add.

int **usb_host_driver_remove** (**struct** usb_host_driver_t **self_p*, **struct** *usb_host_device_driver_t* **driver_p*)
Remove given class/vendor driver from the USB host driver.

Return zero(0) or negative error code.

Parameters

- `self_p`: Initialized driver object.
- `driver_p`: USB device driver to remove.

struct *usb_host_device_t* ***usb_host_device_open** (**struct** usb_host_driver_t **self_p*, int *device*)
Open given device in given driver. Open a device before reading and writing data to it with `usb_host_device_read()` or `usb_host_device_write()`.

Return Opened device or NULL on failure.

Parameters

- `self_p`: Initialized driver.

- `device`: Device to open.

int **usb_host_device_close** (struct *usb_host_driver_t* **self_p*, int *device*)

Close given device in given driver.

Return zero(0) or negative error code.

Parameters

- `self_p`: Initialized driver.
- `device`: Device to close.

ssize_t **usb_host_device_read** (struct *usb_host_device_t* **device_p*, int *endpoint*, void **buf_p*, size_t *size*)

Read data from given endpoint for given device.

Return Number of bytes read or negative error code.

Parameters

- `device_p`: Device to read from.
- `endpoint`: Endpoint to read data from.
- `buf_p`: Buffer to read into.
- `size`: Number of bytes to read.

ssize_t **usb_host_device_write** (struct *usb_host_device_t* **device_p*, int *endpoint*, const void **buf_p*, size_t *size*)

Write data to given endpoint for given device.

Return Number of bytes written or negative error code.

Parameters

- `device_p`: Device to write to.
- `endpoint`: Endpoint to write to.
- `buf_p`: Buffer to write.
- `size`: Number of bytes to write.

ssize_t **usb_host_device_control_transfer** (struct *usb_host_device_t* **device_p*, struct *usb_setup_t* **setup_p*, void **buf_p*, size_t *size*)

Perform a control transfer on endpoint zero(0).

A control transfer can have up to three stages. First the setup stage, then an optional data stage, and at last a status stage.

Return Number of bytes read/written or negative error code.

Parameters

- `device_p`: Device to write to.
- `setup_p`: Setup packet to write.
- `buf_p`: Buffer to read/write. May be NULL if no data shall be transferred.
- `size`: Number of bytes to read/write.

```
int usb_host_device_set_configuration (struct usb_host_device_t *device_p, uint8_t configuration)
```

Set configuration for given device.

Return zero(0) or negative error code.

Parameters

- *device_p*: Device to use.
- *configuration*: Configuration to set.

```
struct usb_host_device_t  
#include <usb_host.h> An USB device as seen by the host.
```

Public Members

```
int id  
int state  
int address  
int vid  
int pid  
char *description_p  
size_t max_packet_size  
uint8_t configuration  
struct usb_descriptor_device_t *dev_p  
struct usb_descriptor_configuration_t *conf_p  
struct usb_host_device_t::@22::@24 usb_host_device_t::descriptor  
struct usb_host_device_t::@22 usb_host_device_t::current  
struct usb_host_driver_t *self_p  
struct usb_pipe_t *pipes[32]  
size_t size  
uint8_t buf[128]  
struct usb_host_device_t::@23 usb_host_device_t::descriptors  
struct usb_host_device_driver_t  
#include <usb_host.h> Used to find a device driver.
```

Public Members

```
int (*supports) (struct usb_host_device_t *)  
int (*enumerate) (struct usb_host_device_t *)  
struct usb_host_device_driver_t *next_p
```

watchdog — Hardware watchdog

Source code: `src/drivers/watchdog.h`, `src/drivers/watchdog.c`

Functions

`int watchdog_module_init (void)`

`int watchdog_start_ms (int timeout)`

Start the watchdog with given timeout. Use `watchdog_kick()` to periodically restart the timer.

Return zero(0) or negative error code.

Parameters

- `timeout`: Watchdog timeout in milliseconds.

`int watchdog_stop (void)`

Stop the watchdog.

Return zero(0) or negative error code.

`int watchdog_kick (void)`

Kick the watchdog. Restarts the watchdog timer with its original timeout given to `watchdog_start_ms()`. The board will be reset if this function is not called before the watchdog timer expires.

Return zero(0) or negative error code.

1.6.3 sync

Thread synchronization refers to the idea that multiple threads are to join up or handshake at a certain point, in order to reach an agreement or commit to a certain sequence of action.

The sync package on [Github](#).

bus — Message bus

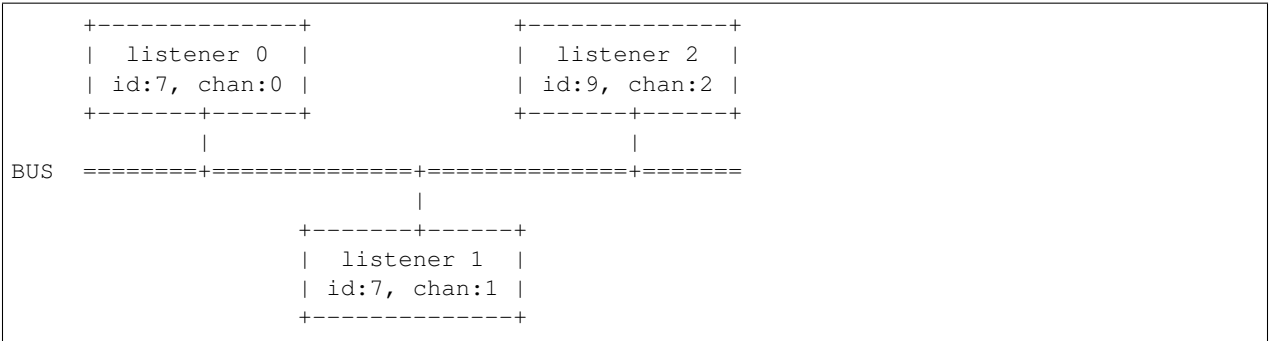
A message bus provides a software-bus abstraction that gathers all the communications between a group of threads over a single shared virtual channel. Messages are transferred on the bus from a sender to one or more attached listeners. The concept is analogous to the bus concept found in computer hardware architecture.

Example

In this example there is a bus with three listeners attached; listener 0, 1 and 2. Listener 0 and 1 are attached to the bus listening for message id 7, and listener 2 for message id 9.

Any thread can write a message to the bus by calling `bus_write()`. If a message with id 7 is written to the bus, both listener 0 and 1 will receive the message. Listener 2 will receive messages with id 9.

Messages are read from the listener channel by the thread that owns the listener.



Source code: [src/sync/bus.h](#), [src/sync/bus.c](#)

Test code: [tst/sync/bus/main.c](#)

Test coverage: [src/sync/bus.c](#)

Functions

int **bus_module_init** (void)

Initialize the bus module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code

int **bus_init** (struct *bus_t* *self_p)

Initialize given bus.

Return zero(0) or negative error code.

Parameters

- self_p: Bus to initialize.

int **bus_listener_init** (struct *bus_listener_t* *self_p, int id, void *chan_p)

Initialize given listener to receive messages with given id, after the listener is attached to the bus. A listener can only receive messages of a single id, though, the same channel may be used in multiple listeners with different ids (if the channel supports it).

Return zero(0) or negative error code.

Parameters

- self_p: Listener to initialize.
- id: Message id to receive.
- chan_p: Channel to receive messages on.

int **bus_attach** (struct *bus_t* *self_p, struct *bus_listener_t* *listener_p)

Attach given listener to given bus. Messages written to the bus will be written to all listeners initialized with the written message id.

Return zero(0) or negative error code.

Parameters

- `self_p`: Bus to attach the listener to.
- `listener_p`: Listener to attach to the bus.

int **bus_detach** (struct *bus_t* *`self_p`, struct *bus_listener_t* *`listener_p`)

Detach given listener from given bus. A detached listener will not receive any messages from the bus.

Return zero(0) or negative error code.

Parameters

- `self_p`: Bus to detach listener from.
- `listener_p`: Listener to detach from the bus.

int **bus_write** (struct *bus_t* *`self_p`, int `id`, const void *`buf_p`, size_t `size`)

Write given message to given bus. All attached listeners to given bus will receive the message.

Return Number of listeners that received the message, or negative error code.

Parameters

- `self_p`: Bus to write the message to.
- `id`: Message identity.
- `buf_p`: Buffer to write to the bus. All listeners with given message id will receive this data.
- `size`: Number of bytes to write.

```
struct bus_t
#include <bus.h>
```

Public Members

```
struct rwlock_t rwlock
struct binary_tree_t listeners
```

```
struct bus_listener_t
```

Public Members

```
struct binary_tree_node_t base
int id
void *chan_p
struct bus_listener_t *next_p
```

chan — Abstract channel communication

Threads often communicate over channels. The producer thread or isr writes data to a channel and the consumer reads it. There may be multiple producers writing to a single channel, but only one consumer is allowed.

In the first example, thread 0 and thread 1 communicate over a channel. thread 0 writes data to the channel and thread 1 reads the written data.

```
+-----+               +-----+
| thread 0 | channel 0 | thread 1 |
|           +-----+           |
| producer |           | consumer |
+-----+               +-----+
```

In the second example, isr 0 and thread 2 communicate over a channel. isr 0 writes data to the channel and thread 2 reads the written data.

```
+-----+               +-----+
| isr 0   | channel 1 | thread 2 |
|         +-----+         |
| producer |           | consumer |
+-----+               +-----+
```

Source code: [src/sync/chan.h](#), [src/sync/chan.c](#)

Test coverage: [src/sync/chan.c](#)

Typedefs

typedef ssize_t (***chan_read_fn_t**) (void *self_p, void *buf_p, size_t size)

typedef ssize_t (***chan_write_fn_t**) (void *self_p, const void *buf_p, size_t size)
Channel write function callback type.

Return Number of written bytes or negative error code.

Parameters

- `self_p`: Channel to write to.
- `buf_p`: Buffer to write.
- `size`: Number of bytes to write.

typedef int (***chan_write_filter_fn_t**) (void *self_p, const void *buf_p, size_t size)
Channel write filter function callback type.

Return true(1) if the buffer shall be written to the channel, otherwise false(0).

Parameters

- `self_p`: Channel to write to.
- `buf_p`: Buffer to write.
- `size`: Number of bytes in buffer.

typedef size_t (*chan_size_fn_t) (void *self_p)
Channel size function callback type.

Return Number of bytes available.

Parameters

- self_p: Channel to get the size of.

Functions

int **chan_module_init** (void)

Initialize the channel module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

int **chan_init** (struct chan_t *self_p, chan_read_fn_t read, chan_write_fn_t write, chan_size_fn_t size)

Initialize given channel with given callbacks. A channel must be initialized before it can be used.

Return zero(0) or negative error code.

Parameters

- self_p: Channel to initialize.
- read: Read function callback. This function must implement the channel read functionality, and will be called when the user reads data from the channel.
- write: Write function callback. This function must implement the channel write functionality, and will be called when the user writes data to the channel.
- size: Size function callback. This function must return the size of the channel. It should return zero(0) if there is no data available in the channel, and otherwise a positive integer.

int **chan_set_write_isr_cb** (struct chan_t *self_p, chan_write_fn_t write_isr_cb)

Set the write isr function callback.

Return zero(0) or negative error code.

Parameters

- self_p: Initialized driver object.
- filter: Write isr function to set.

int **chan_set_write_filter_cb** (struct chan_t *self_p, chan_write_filter_fn_t write_filter_cb)

Set the write filter callback function. The write filter function is called when data is written to the channel, and its return value determines if the data shall be written to the underlying channel implementation, or discarded.

Return zero(0) or negative error code.

Parameters

- self_p: Initialized driver object.
- write_filter_cb: filter Write filter function to set.

```
int chan_set_write_filter_isr_cb (struct chan_t *self_p, chan_write_filter_fn_t
                                write_filter_isr_cb)
```

Set the write isr filter callback function. The write filter function is called when data is written to the channel, and its return value determines if the data shall be written to the underlying channel implementation, or discarded.

Return zero(0) or negative error code.

Parameters

- self_p: Initialized driver object.
- write_filter_isr_cb: filter Write filter function to set.

```
ssize_t chan_read (void *self_p, void *buf_p, size_t size)
```

Read data from given channel. The behaviour of this function depends on the channel implementation. Often, the calling thread will be blocked until all data has been read or an error occurs.

Return Number of read bytes or negative error code.

Parameters

- self_p: Channel to read from.
- buf_p: Buffer to read into.
- size: Number of bytes to read.

```
ssize_t chan_write (void *self_p, const void *buf_p, size_t size)
```

Write data to given channel. The behaviour of this function depends on the channel implementation. Some channel implementations blocks until the receiver has read the data, and some returns immediately.

Return Number of written bytes or negative error code.

Parameters

- self_p: Channel to write to.
- buf_p: Buffer to write.
- size: Number of bytes to write.

```
size_t chan_size (void *self_p)
```

Get the number of bytes available to read from given channel.

Return Number of bytes available.

Parameters

- self_p: Channel to get the size of.

```
ssize_t chan_write_isr (void *self_p, const void *buf_p, size_t size)
```

Write data to given channel from interrupt context or with the system lock taken. The behaviour of this function depends on the channel implementation. Some channel implementations blocks until the receiver has read the data, and some returns immediately.

Return Number of written bytes or negative error code.

Parameters

- self_p: Channel to write to.
- buf_p: Buffer to write.

- `size`: Number of bytes to write.

int **chan_is_polled_isr** (struct *chan_t* **self_p*)

Check if a channel is polled. May only be called from isr or with the system lock taken (see `sys_lock()`).

Return true(1) or false(0).

Parameters

- `self_p`: Channel to check.

int **chan_list_init** (struct *chan_list_t* **list_p*, void **workspace_p*, size_t *size*)

Initialize an empty list of channels. A list is used to wait for data on multiple channel at the same time. When there is data on at least one channel, the poll function returns and the application can read from the channel with data.

Return zero(0) or negative error code.

Parameters

- `list_p`: List to initialize.
- `workspace_p`: Workspace for internal use.
- `size`: Size of the workspace in bytes.

int **chan_list_destroy** (struct *chan_list_t* **list_p*)

Destroy an initialized list of channels.

Return zero(0) or negative error code.

Parameters

- `list_p`: List to destroy.

int **chan_list_add** (struct *chan_list_t* **list_p*, void **chan_p*)

Add given channel to list of channels.

Return zero(0) or negative error code.

Parameters

- `list_p`: List of channels.
- `chan_p`: Channel to add.

int **chan_list_remove** (struct *chan_list_t* **list_p*, void **chan_p*)

Remove given channel from list of channels.

Return zero(0) or negative error code.

Parameters

- `list_p`: List of channels.
- `chan_p`: Channel to remove.

void ***chan_list_poll** (struct *chan_list_t* **list_p*, struct *time_t* **timeout_p*)

Poll given list of channels for events. Blocks until at least one of the channels in the list has data ready to be read or an timeout occurs.

Return Channel with data or NULL on timeout.

Parameters

- `list_p`: List of channels to poll.
- `timeout_p`: Time to wait for data on any channel before a timeout occurs. Set to NULL to wait forever.

void ***chan_poll** (void **chan_p*, struct *time_t* **timeout_p*)

Poll given channel for events. Blocks until the channel has data ready to be read or an timeout occurs.

Return The channel or NULL on timeout.

Parameters

- `chan_p`: Channel to poll.
- `timeout_p`: Time to wait for data on the channel before a timeout occurs. Set to NULL to wait forever.

void ***chan_null** (void)

Get a reference to the null channel. This channel will ignore all written data but return that it was successfully written.

Return The null channel.

ssize_t **chan_read_null** (void **self_p*, void **buf_p*, size_t *size*)

Null channel read function callback. Pass to `chan_init()` if no read function callback is needed for the channel.

Return Always returns -1.

ssize_t **chan_write_null** (void **self_p*, const void **buf_p*, size_t *size*)

Null channel write function callback. Pass to `chan_init()` if no write function callback is needed for the channel.

Return Always returns `size`.

size_t **chan_size_null** (void **self_p*)

Null channel size function callback. Pass to `chan_init()` if no size function callback is needed for the channel.

Return Always returns zero(0).

struct **chan_list_t**

Public Members

struct *chan_t* ****chans_pp**

size_t **max**

size_t **len**

int **flags**

struct chan_t

#include <chan.h> Channel datastructure.

Public Members

chan_read_fn_t **read**

chan_write_fn_t **write**

chan_size_fn_t **size**

chan_write_filter_fn_t **write_filter_cb**

chan_write_fn_t **write_isr**

chan_write_filter_fn_t **write_filter_isr_cb**

struct *thrd_t* ***writer_p**

struct *thrd_t* ***reader_p**

struct *chan_list_t* ***list_p**

event — Event channel

An event channel consists of a 32 bits bitmap, where each bit corresponds to an event state. If the bit is set, the event is active. Since an event only has two states, active and inactive, signalling the same event multiple times will just result in the event to be active. There is no internal counter of how “active” an event is, it’s simply active or inactive.

Source code: [src/sync/event.h](#), [src/sync/event.c](#)

Test code: [tst/sync/event/main.c](#)

Test coverage: [src/sync/event.c](#)

Functions

int **event_init** (**struct** *event_t* **self_p*)

Initialize given event channel.

Return zero(0) or negative error code

Parameters

- *self_p*: Event channel to initialize.

ssize_t **event_read** (**struct** *event_t* **self_p*, void **buf_p*, **size_t** *size*)

Wait for an event to occur in given event mask. This function blocks until at least one of the events in the event mask has been set. When the function returns, given event mask has been overwritten with the events that actually occurred.

Return sizeof(mask) or negative error code.

Parameters

- `self_p`: Event channel object.
- `buf_p`: The mask of events to wait for. When the function returns the mask contains the events that have occurred.
- `size`: Size to read (always `sizeof(mask)`).

`ssize_t event_write (struct event_t *self_p, const void *buf_p, size_t size)`

Write given event(s) to given event channel.

Return `sizeof(mask)` or negative error code.

Parameters

- `self_p`: Event channel object.
- `buf_p`: The mask of events to write.
- `size`: Must always be `sizeof(mask)`.

`ssize_t event_write_isr (struct event_t *self_p, const void *buf_p, size_t size)`

Write given events to the event channel from isr or with the system lock taken (see `sys_lock()`).

Return `sizeof(mask)` or negative error code.

Parameters

- `self_p`: Event channel object.
- `buf_p`: The mask of events to write.
- `size`: Must always be `sizeof(mask)`.

`ssize_t event_size (struct event_t *self_p)`

Checks if there are events active on the event channel.

Return `one(1)` if at least one event is active, otherwise `zero(0)`.

Parameters

- `self_p`: Event channel object.

`struct event_t`

`#include <event.h>`

Public Members

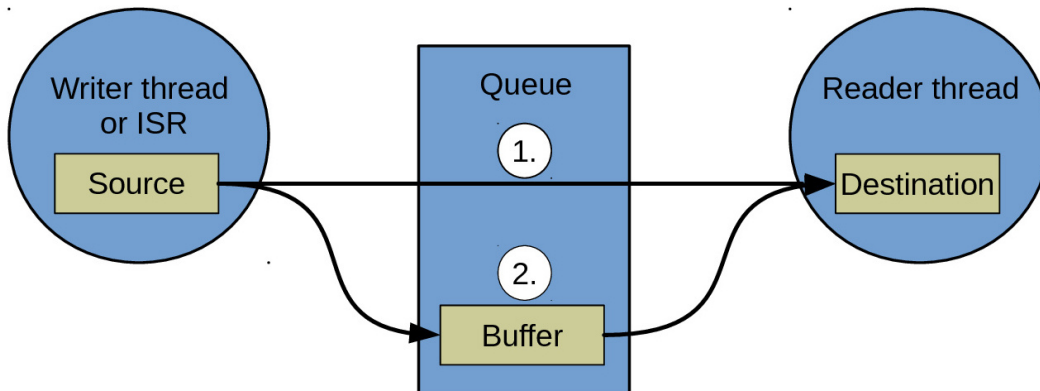
`struct chan_t base`

`uint32_t mask`

queue — Queue channel

The most common channel is the queue. It can be either synchronous or semi-asynchronous. In the synchronous version the writing thread will block until all written data has been read by the reader. In the semi-asynchronous version the writer writes to a buffer within the queue, and only blocks all data does not fit in the buffer. The buffer size is selected by the application when initializing the queue.

The diagram below shows how two threads communicates using a queue. The writer thread writes from its source buffer to the queue. The reader thread reads from the queue to its destination buffer.



The data is either copied directly from the source to the destination buffer (1. in the figure), or via the internal queue buffer (2. in the figure).

1. The reader thread is waiting for data. The writer writes from its source buffer directly to the readers' destination buffer.
2. The reader thread is *not* waiting for data. The writer writes from its source buffer into the queue buffer. Later, the reader reads data from the queue buffer to its destination buffer.

Source code: [src/sync/queue.h](#), [src/sync/queue.c](#)

Test code: [tst/sync/queue/main.c](#)

Test coverage: [src/sync/queue.c](#)

Example code: [examples/queue/main.c](#)

Defines

QUEUE_INIT_DECL (*_name*, *_buf*, *_size*)

Enums

enum queue_state_t

Values:

QUEUE_STATE_INITIALIZED = 0

Queue initialized state.

QUEUE_STATE_RUNNING

Queue running state.

QUEUE_STATE_STOPPED

Queue stopped state.

Functions

int queue_init (**struct queue_t** *self_p, void *buf_p, size_t size)
Initialize given queue.

Return zero(0) or negative error code

Parameters

- self_p: Queue to initialize.
- buf_p: Buffer.
- size: Size of buffer.

int queue_start (**struct queue_t** *self_p)
Start given queue. It is not required to start a queue unless it has been stopped.

Return zero(0) or negative error code.

Parameters

- self_p: Queue to start.

int queue_stop (**struct queue_t** *self_p)
Stop given queue. Any ongoing read and write operations will return with the currently read/written number of bytes. Any read and write operations on a stopped queue will return zero(0).

Return true(1) if a thread was resumed, false(0) if no thread was resumed, or negative error code.

Parameters

- self_p: Queue to stop.

int queue_stop_isr (**struct queue_t** *self_p)
Same as queue_stop() but from isr or with the system lock taken (see sys_lock()).

ssize_t queue_read (**struct queue_t** *self_p, void *buf_p, size_t size)
Read from given queue. Blocks until size bytes has been read.

Return Number of read bytes or negative error code.

Parameters

- self_p: Queue to read from.
- buf_p: Buffer to read to.
- size: Size to read.

ssize_t queue_write (**struct queue_t** *self_p, **const** void *buf_p, size_t size)
Write bytes to given queue. Blocks until size bytes has been written.

Return Number of written bytes or negative error code.

Parameters

- self_p: Queue to write to.
- buf_p: Buffer to write from.
- size: Number of bytes to write.

`ssize_t queue_write_isr (struct queue_t *self_p, const void *buf_p, size_t size)`

Write bytes to given queue from isr or with the system lock taken (see `sys_lock()`). May write less than size bytes.

Return Number of written bytes or negative error code.

Parameters

- `self_p`: Queue to write to.
- `buf_p`: Buffer to write from.
- `size`: Number of bytes to write.

`ssize_t queue_size (struct queue_t *self_p)`

Get the number of bytes currently stored in the queue. May return less bytes than number of bytes stored in the channel.

Return Number of bytes in queue.

Parameters

- `self_p`: Queue.

`ssize_t queue_unused_size (struct queue_t *self_p)`

Get the number of unused bytes in the queue.

Return Number of bytes unused in the queue.

Parameters

- `self_p`: Queue.

`ssize_t queue_unused_size_isr (struct queue_t *self_p)`

Get the number of unused bytes in the queue from isr or with the system lock taken (see `sys_lock()`).

Return Number of bytes unused in the queue.

Parameters

- `self_p`: Queue.

`struct queue_buffer_t`

Public Members

`char *begin_p`

`char *read_p`

`char *write_p`

`char *end_p`

`size_t size`

`struct queue_t`

Public Members

```
struct chan_t base
struct queue_buffer_t buffer
queue_state_t state
char *buf_p
size_t size
size_t left
```

rwlock — Reader-writer lock

An RW lock allows concurrent access for read-only operations, while write operations require exclusive access. This means that multiple threads can read the data in parallel but an exclusive lock is needed for writing or modifying data. When a writer is writing the data, all other writers or readers will be blocked until the writer is finished writing. A common use might be to control access to a data structure in memory that cannot be updated atomically and is invalid (and should not be read by another thread) until the update is complete.

Source code: [src/sync/rwlock.h](#), [src/sync/rwlock.c](#)

Test code: [tst/sync/rwlock/main.c](#)

Test coverage: [src/sync/rwlock.c](#)

Functions

int **rwlock_module_init** (void)

Initialize the reader-writer lock module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code

int **rwlock_init** (struct *rwlock_t* *self_p)

Initialize given reader-writer lock object.

Return zero(0) or negative error code.

Parameters

- self_p: Reader-writer lock to initialize.

int **rwlock_reader_take** (struct *rwlock_t* *self_p)

Take given reader-writer lock. Multiple threads can have the reader lock at the same time.

Return zero(0) or negative error code.

Parameters

- self_p: Reader-writer lock to take.

int **rwlock_reader_give** (struct *rwlock_t* *self_p)

Give given reader-writer lock.

Return zero(0) or negative error code.

Parameters

- self_p: Reader-writer lock give.

int **rwlock_reader_give_isr** (struct *rwlock_t* *self_p)

Give given reader-writer lock from isr or with the system lock taken.

Return zero(0) or negative error code.

Parameters

- self_p: Reader-writer lock to give.

int **rwlock_writer_take** (struct *rwlock_t* *self_p)

Take given reader-writer lock as a writer. Only one thread can have the lock at a time, including both readers and writers.

Return zero(0) or negative error code.

Parameters

- self_p: Reader-writer lock to take.

int **rwlock_writer_give** (struct *rwlock_t* *self_p)

Give given reader-writer lock.

Return zero(0) or negative error code.

Parameters

- self_p: Reader-writer lock to give.

int **rwlock_writer_give_isr** (struct *rwlock_t* *self_p)

Give given reader-writer lock from isr or with the system lock taken.

Return zero(0) or negative error code.

Parameters

- self_p: Reader-writer lock to give.

struct **rwlock_t**

#include <rwlock.h>

Public Members

int **number_of_readers**

int **number_of_writers**

volatile struct rwlock_elem_t ***readers_p**

volatile struct rwlock_elem_t ***writers_p**

sem — Counting semaphores

The semaphore is a synchronization primitive used to protect a shared resource. A semaphore counts the number of resources taken, and suspends threads when the maximum number of resources are taken. When a resource becomes available, a suspended thread is resumed.

A semaphore initialized with *count_max* one(1) is called a binary semaphore. A binary semaphore can only be taken by one thread at a time and can be used to signal that an event has occurred. That is, *sem_give()* may be called multiple times and the semaphore resource count will remain at zero(0) until *sem_take()* is called.

Source code: [src/sync/sem.h](#), [src/sync/sem.c](#)

Test code: [tst/sync/sem/main.c](#)

Test coverage: [src/sync/sem.c](#)

Defines

SEM_INIT_DECL (name, _count, _count_max)

Functions

int **sem_module_init** (void)

Initialize the semaphore module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code

int **sem_init** (struct *sem_t* *self_p, int count, int count_max)

Initialize given semaphore object. Maximum count is the number of resources that can be taken at any given moment.

Return zero(0) or negative error code.

Parameters

- self_p: Semaphore to initialize.
- count: Initial taken resource count. Set the initial count to the same value as count_max to initialize the semaphore with all resources taken.
- count_max: Maximum number of resources that can be taken at any given moment.

int **sem_take** (struct *sem_t* *self_p, struct *time_t* *timeout_p)

Take given semaphore. If the semaphore count is zero the calling thread will be suspended until count is incremented by *sem_give()*.

Return zero(0) or negative error code.

Parameters

- self_p: Semaphore to get.
- timeout_p: Timeout.

int **sem_give** (struct *sem_t* **self_p*, int *count*)

Give given count to given semaphore. Any suspended thread waiting for this semaphore, in `sem_take()`, is resumed. This continues until the semaphore count becomes zero or there are no threads in the suspended list.

Giving a count greater than the currently taken count is allowed and results in all resources available. This is especially useful for binary semaphores where `sem_give()` is often called more often than `sem_take()`.

Return zero(0) or negative error code.

Parameters

- *self_p*: Semaphore to give count to.
- *count*: Count to give.

int **sem_give_isr** (struct *sem_t* **self_p*, int *count*)

Give given count to given semaphore from isr or with the system lock taken.

Return zero(0) or negative error code.

Parameters

- *self_p*: Semaphore to give count to.
- *count*: Count to give.

struct **sem_t**

Public Members

int **count**

Number of used resources.

int **count_max**

Maximum number of resources.

struct *sem_elem_t* ***head_p**

Wait list.

1.6.4 filesystems

File systems and file system like frameworks.

The filesystems package on [Github](#).

fat16 — FAT16 filesystem

File Allocation Table (FAT) is a computer file system architecture and a family of industry-standard file systems utilizing it. The FAT file system is a legacy file system which is simple and robust. It offers good performance even in light-weight implementations, but cannot deliver the same performance, reliability and scalability as some modern file systems. It is, however, supported for compatibility reasons by nearly all currently developed operating systems for personal computers and many mobile devices and embedded systems, and thus is a well-suited format for data exchange between computers and devices of almost any type and age from 1981 up to the present.

Example

Here is the pseudo-code for mounting a file system, performing file operations and unmounting the file system.

All function arguments are omitted in this example.

```
/* Mount the file system. This is normally done once when the
   application starts. */
fat16_init();
fat16_mount();

/* Perform file operations. */
fat16_file_open();
fat16_file_read();
fat16_file_close();

fat16_file_open();
fat16_file_write();
fat16_file_close();

/* Unmount the file system when it is no long needed. Normally when
   the application stops. */
fat16_unmount();
```

Source code: `src/filesystems/fat16.h`, `src/filesystems/fat16.c`

Test code: `tst/filesystems/fat16/main.c`

Test coverage: `src/filesystems/fat16.c`

Example code: `examples/fat16/main.c`

Defines

FAT16_SEEK_SET

FAT16_SEEK_CUR

The offset is relative to the current position indicator.

FAT16_SEEK_END

The offset is relative to the end of the file.

FAT16_EOF

End of file indicator.

O_READ

Open for reading.

O_RDONLY

Same as O_READ.

O_WRITE

Open for write.

O_WRONLY

Same as O_WRITE.

O_RDONLY

Open for reading and writing.

O_APPEND

The file position indicator shall be set to the end of the file prior to each write.

O_SYNC

Synchronous writes.

O_CREAT

Create the file if non-existent.

O_EXCL

If O_CREAT and O_EXCL are set, file open shall fail if the file exists.

O_TRUNC

Truncate the file to zero length.

DIR_ATTR_READ_ONLY

File is read-only.

DIR_ATTR_HIDDEN

File should hidden in directory listings.

DIR_ATTR_SYSTEM

Entry is for a system file.

DIR_ATTR_VOLUME_ID

Directory entry contains the volume label.

DIR_ATTR_DIRECTORY

Entry is for a directory.

DIR_ATTR_ARCHIVE

Old DOS archive bit for backup support.

Typedefs

```
typedef ssize_t (*fat16_read_t) (void *arg_p, void *dst_p, uint32_t src_block)
```

Block read function callback.

```
typedef ssize_t (*fat16_write_t) (void *arg_p, uint32_t dst_block, const void *src_p)
```

Block write function callback.

```
typedef uint16_t fat_t
```

A FAT entry.

Functions

```
int fat16_init (struct fat16_t *self_p, fat16_read_t read, fat16_write_t write, void *arg_p, unsigned int
                partition)
```

Initialize a FAT16 volume.

Return zero(0) or negative error code.

Parameters

- `self_p`: FAT16 object to initialize.
- `read`: Callback function used to read blocks of data.

- `write`: Callback function used to write blocks of data.
- `arg_p`: Argument passed as the first argument to `read()` and `write()`.
- `partition`: Partition to be used. Legal values for a partition are 1-4 to use the corresponding partition on a device formatted with a MBR, Master Boot Record, or zero if the device is formatted as a super floppy with the FAT boot sector in block zero.

int **fat16_mount** (**struct** *fat16_t* **self_p*)
Mount given FAT16 volume.

Return zero(0) or negative error code.

Parameters

- `self_p`: FAT16 object.

int **fat16_unmount** (**struct** *fat16_t* **self_p*)
Unmount given FAT16 volume.

Return zero(0) or negative error code.

Parameters

- `self_p`: FAT16 object.

int **fat16_format** (**struct** *fat16_t* **self_p*)
Create an empty FAT16 file system on the device.

Parameters

- `self_p`: FAT16 object.

int **fat16_print** (**struct** *fat16_t* **self_p*, void **chan_p*)
Print volume information to given channel.

Return zero(0) or negative error code.

Parameters

- `self_p`: FAT16 object.
- `chan_p`: Output channel.

int **fat16_file_open** (**struct** *fat16_t* **self_p*, **struct** *fat16_file_t* **file_p*, **const** char **path_p*, int *oflag*)
Open a file by file path and mode flags.

Return zero(0) or negative error code.

Parameters

- `self_p`: FAT16 object.
- `file_p`: File object to be initialized.
- `path_p`: A valid 8.3 DOS name for a file path.
- `oflag`: mode of file open (create, read, write, etc).

int **fat16_file_close** (**struct** *fat16_file_t* **file_p*)
Close a file and force cached data and directory information to be written to the media.

Return zero(0) or negative error code.

Parameters

- `file_p`: File object.

`ssize_t fat16_file_read(struct fat16_file_t *file_p, void *buf_p, size_t size)`

Read data to given buffer with given size from the file.

Return Number of bytes read or EOF(-1).

Parameters

- `file_p`: File object.
- `buf_p`: Buffer to read into.
- `size`: number of bytes to read.

`ssize_t fat16_file_write(struct fat16_file_t *file_p, const void *buf_p, size_t size)`

Write data from buffer with given size to the file.

Return Number of bytes written or EOF(-1).

Parameters

- `file_p`: File object.
- `buf_p`: Buffer to write.
- `size`: number of bytes to write.

`int fat16_file_seek(struct fat16_file_t *file_p, int pos, int whence)`

Sets the file's read/write position relative to mode.

Return zero(0) or negative error code.

Parameters

- `file_p`: File object.
- `pos`: New position in bytes from given mode.
- `whence`: Absolute, relative or from end.

`ssize_t fat16_file_tell(struct fat16_file_t *file_p)`

Return current position in the file.

Return Current position or negative error code.

Parameters

- `file_p`: File object.

`int fat16_file_truncate(struct fat16_file_t *file_p, size_t size)`

Truncate given file to a size of precisely `size` bytes.

If the file previously was larger than this size, the extra data is lost. If the file previously was shorter, it is extended, and the extended part reads as null bytes (`'\0'`).

Return zero(0) or negative error code.

Parameters

- `file_p`: File object.
- `size`: New size of the file in bytes.

`ssize_t fat16_file_size (struct fat16_file_t *file_p)`
Return number of bytes in the file.

Return File size in bytes or negative error code.

Parameters

- `file_p`: File object.

`int fat16_file_sync (struct fat16_file_t *file_p)`
Causes all modified data and directory fields to be written to the storage device.

Return zero(0) or negative error code.

Parameters

- `file_p`: File object.

`int fat16_dir_open (struct fat16_t *self_p, struct fat16_dir_t *dir_p, const char *path_p, int oflag)`
Open a directory by directory path and mode flags.

Return zero(0) or negative error code.

Parameters

- `self_p`: FAT16 object.
- `dir_p`: Directory object to be initialized.
- `path_p`: A valid 8.3 DOS name for a directory path.
- `oflag`: mode of the directory to open (create, read, etc).

`int fat16_dir_close (struct fat16_dir_t *dir_p)`
Close given directory.

Return zero(0) or negative error code.

Parameters

- `dir_p`: Directory object.

`int fat16_dir_read (struct fat16_dir_t *dir_p, struct fat16_dir_entry_t *entry_p)`
Read the next file or directory within the opened directory.

Return true(1) if an entry was read or false(0) if no entry could be read, otherwise negative error code.

Parameters

- `dir_p`: Directory object.
- `entry_p`: Read entry.

`int fat16_stat (struct fat16_t *self_p, const char *path_p, struct fat16_stat_t *stat_p)`
Gets file status by path.

Return zero(0) or negative error code.

Parameters

- `self_p`: The file system struct.
- `path_p`: The path of the file to stat.
- `stat_p`: The stat struct to populate.

Variables

struct *dir_t* PACKED

union *fat16_time_t*

#include <fat16.h> FAT Time Format. A FAT directory entry time stamp is a 16-bit field that has a granularity of 2 seconds. Here is the format (bit 0 is the LSB of the 16-bit word, bit 15 is the MSB of the 16-bit word).

Bits 0-4: 2-second count, valid value range 0-29 inclusive (0-58 seconds). Bits 5-10: Minutes, valid value range 0-59 inclusive. Bits 11-15: Hours, valid value range 0-23 inclusive.

The valid time range is from Midnight 00:00:00 to 23:59:58.

Public Members

`uint16_t as_uint16`

`uint16_t seconds`

`uint16_t minutes`

`uint16_t hours`

struct *fat16_time_t*::@25 *fat16_time_t*::bits

union *fat16_date_t*

#include <fat16.h> FAT date representation support Date Format. A FAT directory entry date stamp is a 16-bit field that is basically a date relative to the MS-DOS epoch of 01/01/1980. Here is the format (bit 0 is the LSB of the 16-bit word, bit 15 is the MSB of the 16-bit word):

Bits 0-4: Day of month, valid value range 1-31 inclusive. Bits 5-8: Month of year, 1 = January, valid value range 1-12 inclusive. Bits 9-15: Count of years from 1980, valid value range 0-127 inclusive (1980-2107).

Public Members

`uint16_t as_uint16`

`uint16_t day`

`uint16_t month`

`uint16_t year`

struct *fat16_date_t*::@26 *fat16_date_t*::bits

struct *part_t*

#include <fat16.h> MBR partition table entry. A partition table entry for a MBR formatted storage device. The MBR partition table has four entries.

Public Members

uint8_t **boot**

Boot Indicator. Indicates whether the volume is the active partition. Legal values include: 0x00. Do not use for booting. 0x80 Active partition.

uint8_t **begin_head**

Head part of Cylinder-head-sector address of the first block in the partition. Legal values are 0-255. Only used in old PC BIOS.

unsigned **begin_sector**

Sector part of Cylinder-head-sector address of the first block in the partition. Legal values are 1-63. Only used in old PC BIOS.

unsigned **begin_cylinder_high**

High bits cylinder for first block in partition.

uint8_t **begin_cylinder_low**

Combine beginCylinderLow with beginCylinderHigh. Legal values are 0-1023. Only used in old PC BIOS.

uint8_t **type**

Partition type. See defines that begin with PART_TYPE_ for some Microsoft partition types.

uint8_t **end_head**

head part of cylinder-head-sector address of the last sector in the partition. Legal values are 0-255. Only used in old PC BIOS.

unsigned **end_sector**

Sector part of cylinder-head-sector address of the last sector in the partition. Legal values are 1-63. Only used in old PC BIOS.

unsigned **end_cylinder_high**

High bits of end cylinder

uint8_t **end_cylinder_low**

Combine endCylinderLow with endCylinderHigh. Legal values are 0-1023. Only used in old PC BIOS.

uint32_t **first_sector**

Logical block address of the first block in the partition.

uint32_t **total_sectors**

Length of the partition, in blocks.

struct **bp_b_t**

#include <fat16.h> BIOS parameter block; The BIOS parameter block describes the physical layout of a FAT volume.

Public Members

uint16_t **bytes_per_sector**

Count of bytes per sector. This value may take on only the following values: 512, 1024, 2048 or 4096

uint8_t **sectors_per_cluster**

Number of sectors per allocation unit. This value must be a power of 2 that is greater than 0. The legal values are 1, 2, 4, 8, 16, 32, 64, and 128.

uint16_t **reserved_sector_count**

Number of sectors before the first FAT. This value must not be zero.

uint8_t fat_count

The count of FAT data structures on the volume. This field should always contain the value 2 for any FAT volume of any type.

uint16_t root_dir_entry_count

For FAT12 and FAT16 volumes, this field contains the count of 32-byte directory entries in the root directory. For FAT32 volumes, this field must be set to 0. For FAT12 and FAT16 volumes, this value should always specify a count that when multiplied by 32 results in a multiple of bytesPerSector. FAT16 volumes should use the value 512.

uint16_t total_sectors_small

This field is the old 16-bit total count of sectors on the volume. This count includes the count of all sectors in all four regions of the volume. This field can be 0; if it is 0, then totalSectors32 must be non-zero. For FAT32 volumes, this field must be 0. For FAT12 and FAT16 volumes, this field contains the sector count, and totalSectors32 is 0 if the total sector count fits (is less than 0x10000).

uint8_t media_type

This dates back to the old MS-DOS 1.x media determination and is no longer usually used for anything. 0xf8 is the standard value for fixed (non-removable) media. For removable media, 0xf0 is frequently used. Legal values are 0xf0 or 0xf8-0xff.

uint16_t sectors_per_fat

Count of sectors occupied by one FAT on FAT12/FAT16 volumes. On FAT32 volumes this field must be 0, and sectorsPerFat32 contains the FAT size count.

uint16_t sectors_per_track

Sectors per track for interrupt 0x13. Not used otherwise.

uint16_t head_count

Number of heads for interrupt 0x13. Not used otherwise.

uint32_t hiddden_sectors

Count of hidden sectors preceding the partition that contains this FAT volume. This field is generally only relevant for media visible on interrupt 0x13.

uint32_t total_sectors_large

This field is the new 32-bit total count of sectors on the volume. This count includes the count of all sectors in all four regions of the volume. This field can be 0; if it is 0, then totalSectors16 must be non-zero.

struct fbs_t

#include <fat16.h> Boot sector for a FAT16 or FAT32 volume.

Public Members**uint8_t jmp_to_boot_code[3]**

X86 jmp to boot program

char oem_name[8]

Informational only - don't depend on it

struct bpb_t bpb

BIOS Parameter Block

uint8_t drive_number

For int0x13 use value 0x80 for hard drive

uint8_t reserved1

Used by Windows NT - should be zero for FAT

`uint8_t boot_signature`
0x29 if next three fields are valid

`uint32_t volume_serial_number`
Usually generated by combining date and time

`char volume_label[11]`
Should match volume label in root dir

`char file_system_type[8]`
Informational only - don't depend on it

`uint8_t boot_code[448]`
X86 boot code

`uint16_t boot_sector_sig`
Must be 0x55AA

struct mbr_t

#include <fat16.h> Master Boot Record. The first block of a storage device that is formatted with a MBR.

Public Members

`uint8_t codeArea[440]`
Code Area for master boot program.

`uint32_t diskSignature`
Optional WindowsNT disk signature. May contain more boot code.

`uint16_t usuallyZero`
Usually zero but may be more boot code.

struct *part_t* `part[4]`
Partition tables.

`uint16_t mbr_sig`
First MBR signature byte. Must be 0x55

struct dir_t

#include <fat16.h> FAT short directory entry. Short means short 8.3 name, not the entry size.

Public Members

`uint8_t name[11]`
Short 8.3 name. The first eight bytes contain the file name with blank fill. The last three bytes contain the file extension with blank fill.

`uint8_t attributes`
Entry attributes. The upper two bits of the attribute byte are reserved and should always be set to 0 when a file is created and never modified or looked at after that. See defines that begin with `DIR_ATT_`.

`uint8_t reserved1`
Reserved for use by Windows NT. Set value to 0 when a file is created and never modify or look at it after that.

`uint8_t creation_time_tenths`
The granularity of the seconds part of creationTime is 2 seconds so this field is a count of tenths of a second and its valid value range is 0-199 inclusive. (WHG note - seems to be hundredths)

`uint16_t creation_time`

Time file was created.

`uint16_t creation_date`

Date file was created.

`uint16_t last_access_date`

Last access date. Note that there is no last access time, only a date. This is the date of last read or write. In the case of a write, this should be set to the same date as `lastWriteDate`.

`uint16_t first_cluster_high`

High word of this entry's first cluster number (always 0 for a FAT12 or FAT16 volume).

`uint16_t last_write_time`

Time of last write. File creation is considered a write.

`uint16_t last_write_date`

Date of last write. File creation is considered a write.

`uint16_t first_cluster_low`

Low word of this entry's first cluster number.

`uint32_t file_size`

32-bit unsigned holding this file's size in bytes.

`union fat16_cache16_t`

Public Members

`uint8_t data[512]`

`fat_t fat[256]`

`struct dir_t dir[16]`

`struct mbr_t mbr`

`struct fbs_t fbs`

`struct fat16_cache_t`

Public Members

`uint32_t block_number`

`uint8_t dirty`

`uint32_t mirror_block`

`union fat16_cache16_t buffer`

`struct fat16_t`

Public Members

`fat16_read_t read`

`fat16_write_t write`

`void *arg_p`

```
    unsigned int partition
    uint8_t fat_count
    uint8_t blocks_per_cluster
    uint16_t root_dir_entry_count
    fat_t blocks_per_fat
    fat_t cluster_count
    uint32_t volume_start_block
    uint32_t fat_start_block
    uint32_t root_dir_start_block
    uint32_t data_start_block
    struct fat16_cache_t cache
struct fat16_file_t
```

Public Members

```
struct fat16_t *fat16_p
uint8_t flags
int16_t dir_entry_block
int16_t dir_entry_index
fat_t first_cluster
size_t file_size
fat_t cur_cluster
size_t cur_position
struct fat16_dir_t
```

Public Members

```
int16_t root_index
struct fat16_file_t file
struct fat16_dir_entry_t
```

Public Members

```
char name[256]
int is_dir
size_t size
struct date_t latest_mod_date
struct fat16_stat_t
```

Public Members

size_t **size**

int **is_dir**

fs — Debug file system

The debug file system is not really a file system, but rather a file system like tree of commands, counters, parameters, and “real” file systems.

- A command is a file path mapped to a function callback. The callback is invoked when its path is passed to the `fs_call()` function. Commands are registered into the debug file system by a call to `fs_command_register()`.
- A counter is a file path mapped to a 64 bit value. The value can be incremented and read by the application. Counters are registered into the debug file system by a call to `fs_counter_register()`.
- A parameter is file path mapped to a value stored in ram that can be easily read and modified by the user from a shell. Parameters are registered into the debug file system by a call to `fs_parameter_register()`.
- A “real” file system is a file path, or mount point, mapped to a file system instance. The debug file system has a file access interface. The purpose of this interface is to have a common file access interface, independent of the underlying file systems interface. File systems are registered into the debug file system by a call to `fs_filesystem_register()`.

Debug file system commands

The debug file system module itself registers seven commands, all located in the directory `filesystems/fs/`.

Command	Description
<code>filesystems/list</code>	Print a list of all registered file systems.
<code>list [<folder>]</code>	Print a list of all files and folders in given folder.
<code>read <file></code>	Read from given file.
<code>write <file> <data></code>	Create and write to a file. Overwrites existing files.
<code>append <file> <data></code>	Append data to an existing file.
<code>counters/list</code>	Print a list of all registered counters.
<code>counters/reset</code>	Rest all counters to zero.
<code>parameters/list</code>	Print a list of all registered parameters.

Example output from the shell:

```
$ filesystems/fs/filesystems/list
MOUNT-POINT      MEDIUM  TYPE      AVAILABLE  SIZE  USAGE
/tmp              ram      fat16      54K        64K   14%
/home/erik        sd       fat16      1.9G       2G    5%
/etc              flash    spiffs     124K       128K   3%
$ filesystems/fs/write tmp/foo.txt "Hello "
$ filesystems/fs/append tmp/foo.txt world!
$ filesystems/fs/read tmp/foo.txt
Hello world!
$ filesystems/fs/list tmp
xxxx-xx-xx xx-xx      12 foo.txt
$ filesystems/fs/counters/list
NAME                                     VALUE
/your/counter                0000000000000034
```

/my/counter	0000000000000002
\$ filesystems/fs/counters/reset	
\$ filesystems/fs/counters/list	
NAME	VALUE
/your/counter	0000000000000000
/my/counter	0000000000000000
\$ filesystems/fs/parameters/list	
NAME	VALUE
/foo/bar	-2

Source code: [src/filesystems/fs.h](#), [src/filesystems/fs.c](#)

Test code: [tst/filesystems/fs/main.c](#)

Test coverage: [src/filesystems/fs.c](#)

Defines

FS_SEEK_SET

FS_SEEK_CUR

The offset is relative to the current position indicator.

FS_SEEK_END

The offset is relative to the end of the file.

FS_READ

Open for reading.

FS_WRITE

Open for write.

FS_RDWR

Open for reading and writing.

FS_APPEND

The file position indicator shall be set to the end of the file prior to each write.

FS_SYNC

Synchronous writes.

FS_CREAT

Create the file if non-existent.

FS_EXCL

If FS_CREAT and FS_EXCL are set, file open shall fail if the file exists.

FS_TRUNC

Truncate the file to zero length.

FS_TYPE_FILE

FS_TYPE_DIR

FS_TYPE_HARD_LINK

FS_TYPE_SOFT_LINK

Typedefs

```
typedef int (*fs_callback_t)(int argc, const char *argv[], void *out_p, void *in_p, void *arg_p, void
                             *call_arg_p)
```

Command callback prototype.

Return zero(0) or negative error code.

Parameters

- `argc`: Number of arguments in `argv`.
- `argv`: An array of arguments.
- `out_p`: Output channel.
- `in_p`: Input channel.
- `arg_p`: Argument passed to the init function of given command.
- `call_arg_p`: Argument passed to the `fs_call` function.

```
typedef int (*fs_parameter_set_callback_t)(void *value_p, const char *src_p)
```

Parameter setter callback prototype.

Return zero(0) or negative error code.

Parameters

- `value_p`: Buffer the new value should be written to.
- `src_p`: Value to set as a string.

```
typedef int (*fs_parameter_print_callback_t)(void *chout_p, void *value_p)
```

Parameter printer callback prototype.

Return zero(0) or negative error code.

Parameters

- `chout_p`: Channel to write the formatted value to.
- `value_p`: Value to format and print to the output channel.

Enums

```
enum fs_type_t
```

Values:

```
fs_type_fat16_t = 0
```

```
fs_type_spiffs_t
```

Functions

```
int fs_module_init(void)
```

Initialize the file system module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

int **fs_call** (char **command_p*, void **chin_p*, void **chout_p*, void **arg_p*)

Call given file system command with given input and output channels. Quote an argument if it contains spaces, otherwise it is parsed as multiple arguments. Any quotation mark in an argument string must be escaped with a backslash (\), otherwise it is interpreted as a string quotation mask.

Return zero(0) or negative error code.

Parameters

- *command_p*: Command string to call. The command string will be modified by this function, so don't use it after this function returns.
- *chin_p*: Input channel.
- *chout_p*: Output channel.
- *arg_p*: User argument passed to the command callback function as *call_arg_p*.

int **fs_open** (struct *fs_file_t* **self_p*, const char **path_p*, int *flags*)

Open a file by file path and mode flags. File operations are permitted after the file has been opened.

The path can be either absolute or relative. It's an absolute path if it starts with a forward slash /, and relative otherwise. Relative paths are relative to the current working directory, given by the thread environment variable CWD.

Return zero(0) or negative error code.

Parameters

- *self_p*: File object to be initialized.
- *path_p*: Path of the file to open. The path can be absolute or relative.
- *flags*: Mode of file open. A combination of FS_READ, FS_RDONLY, FS_WRITE, FS_WRONLY, FS_RDWR, FS_APPEND, FS_SYNC, FS_CREAT, FS_EXCL and FS_TRUNC.

int **fs_close** (struct *fs_file_t* **self_p*)

Close given file. No file operations are permitted on a closed file.

Return zero(0) or negative error code.

Parameters

- *self_p*: Initialized file object.

ssize_t **fs_read** (struct *fs_file_t* **self_p*, void **dst_p*, size_t *size*)

Read from given file into given buffer.

Return Number of bytes read or negative error code.

Parameters

- *self_p*: Initialized file object.
- *dst_p*: Buffer to read data into.
- *size*: Number of bytes to read.

`ssize_t fs_read_line (struct fs_file_t *self_p, void *dst_p, size_t size)`

Read one line from given file into given buffer. The function reads one character at a time from given file until the destination buffer is full, a newline `\n` is found or end of file is reached.

Return If a line was found the number of bytes read not including the null-termination is returned. If the destination buffer becomes full before a newline character, the destination buffer size is returned. Otherwise a negative error code is returned.

Parameters

- `self_p`: Initialized file object.
- `dst_p`: Buffer to read data into. Should fit the whole line and null-termination.
- `size`: Size of the destination buffer.

`ssize_t fs_write (struct fs_file_t *self_p, const void *src_p, size_t size)`

Write from given buffer into given file.

Return Number of bytes written or negative error code.

Parameters

- `self_p`: Initialized file object.
- `dst_p`: Buffer to write.
- `size`: Number of bytes to write.

`int fs_seek (struct fs_file_t *self_p, int offset, int whence)`

Sets the file's read/write position relative to whence.

Return zero(0) or negative error code.

Parameters

- `self_p`: Initialized file object.
- `offset`: New position in bytes from given whence.
- `whence`: Absolute (`FS_SEEK_SET`), relative (`FS_SEEK_CUR`) or from end (`FS_SEEK_END`).

`ssize_t fs_tell (struct fs_file_t *self_p)`

Return current position in the file.

Return Current position or negative error code.

Parameters

- `self_p`: Initialized file object.

`int fs_dir_open (struct fs_dir_t *dir_p, const char *path_p, int oflag)`

Open a directory by directory path and mode flags.

Return zero(0) or negative error code.

Parameters

- `dir_p`: Directory object to be initialized.
- `path_p`: A valid path name for a directory path.
- `oflag`: mode of the directory to open (create, read, etc).

int **fs_dir_close** (struct *fs_dir_t* **dir_p*)
Close given directory.

Return zero(0) or negative error code.

Parameters

- *dir_p*: Directory object.

int **fs_dir_read** (struct *fs_dir_t* **dir_p*, struct *fs_dir_entry_t* **entry_p*)
Read the next file or directory within the opened directory.

Return true(1) if an entry was read or false(0) if no entry could be read, otherwise negative error code.

Parameters

- *dir_p*: Directory object.
- *entry_p*: Read entry.

int **fs_remove** (const char **path_p*)
Remove file by given path.

Return zero(0) or negative error code.

Parameters

- *path_p*: The path of the file to remove.

int **fs_stat** (const char **path_p*, struct *fs_stat_t* **stat_p*)
Gets file status by path.

Return zero(0) or negative error code.

Parameters

- *path_p*: The path of the file to stat.
- *stat_p*: The stat struct to populate.

int **fs_mkdir** (const char **path_p*)
Create a directory with given path.

Return zero(0) or negative error code.

Parameters

- *path_p*: The path of the directory to create.

int **fs_format** (const char **path_p*)
Format file system at given path.

Return zero(0) or negative error code.

Parameters

- *path_p*: The path to the root of the file system to format. All data in the file system will be deleted.

int **fs_ls** (const char **path_p*, const char **filter_p*, void **chout_p*)
List files and folders in given path. Optionally with given filter. The list is written to the output channel.

Return zero(0) or negative error code.

Parameters

- `path_p`: Directory to list.
- `filter_p`: Filter out files and folders.
- `chout_p`: Output chan.

int **fs_list** (const char **path_p*, const char **filter_p*, void **chout_p*)

List files (callbacks) and directories in given path. Optionally with given filter. The list is written to the output channel.

Return zero(0) or negative error code.

Parameters

- `path_p`: Directory to list.
- `filter_p`: Filter out files and folders.
- `chout_p`: Output chan.

int **fs_auto_complete** (char **path_p*)

Auto-complete given path.

Return ≥ 1 if completion happened. Number of autocompleted characters added to the path. 0 if no completion happend, or negative error code.

Parameters

- `path_p`: Absolute or relative path to auto-complete.

void **fs_split** (char **buf_p*, char ***path_pp*, char ***cmd_pp*)

Split buffer into path and command inplace.

Return zero(0) or negative error code.

Parameters

- `buf_p`: Buffer to split.
- `path_pp`: Path or NULL if no path was found.
- `cmd_pp`: Command or empty string.

void **fs_merge** (char **path_p*, char **cmd_p*)

Merge path and command previously split using `fs_split()`.

Return zero(0) or negative error code.

Parameters

- `path_p`: Path from spilt.
- `cmd_p`: Command from split.

int **fs_filesystem_init_fat16** (struct *fs_filesystem_t* **self_p*, const char **name_p*, struct *fat16_t* **fat16_p*)

Initialize given FAT16 file system.

Return zero(0) or negative error code.

Parameters

- `self_p`: File system to initialize.
- `name_p`: Path to register.
- `fat16_p`: File system pointer.

```
int fs_filesystem_init_spiffs (struct fs_filesystem_t *self_p, const char *name_p, struct spiffs_t
                               *spiffs_p, struct fs_filesystem_spiffs_config_t *config_p)
```

Initialize given SPIFFS file system.

Return zero(0) or negative error code.

Parameters

- `self_p`: File system to initialize.
- `name_p`: Path to register.
- `spiffs_p`: File system pointer.
- `config_p`: File system configuration.

```
int fs_filesystem_register (struct fs_filesystem_t *self_p)
```

Register given file system. Use the functions `fs_open()`, `fs_read()`, `fs_write()`, `fs_close()`, `fs_seek()`, `fs_tell()` and `fs_read_line()` to access files in a registered file system.

Return zero(0) or negative error code.

Parameters

- `self_p`: File system to register.

```
int fs_filesystem_deregister (struct fs_filesystem_t *self_p)
```

Deregister given file system.

Return zero(0) or negative error code.

Parameters

- `self_p`: File system to deregister.

```
int fs_command_init (struct fs_command_t * self_p, const FAR char * path_p, fs_callback_t
```

Initialize given command.

Return zero(0) or negative error code.

Parameters

- `self_p`: Command to initialize.
- `path_p`: Path to register.
- `callback`: Command callback function.
- `arg_p`: Callback argument.

```
int fs_command_register (struct fs_command_t *command_p)
```

Register given command. Registered commands are called by the function `fs_call()`.

Return zero(0) or negative error code.

Parameters

- `command_p`: Command to register.

`int fs_command_deregister (struct fs_command_t *command_p)`
Deregister given command.

Return zero(0) or negative error code.

Parameters

- `command_p`: Command to deregister.

`int fs_counter_init (struct fs_counter_t *self_p, const FAR char * path_p, uint64_t value)`
Initialize given counter.

Return zero(0) or negative error code.

Parameters

- `self_p`: Counter to initialize.
- `path_p`: Path to register.
- `value`: Initial value of the counter.

`int fs_counter_increment (struct fs_counter_t *self_p, uint64_t value)`
Increment given counter.

Return zero(0) or negative error code.

Parameters

- `self_p`: Command to initialize.
- `value`: Increment value.

`int fs_counter_register (struct fs_counter_t *counter_p)`
Register given counter.

Return zero(0) or negative error code.

Parameters

- `counter_p`: Counter to register.

`int fs_counter_deregister (struct fs_counter_t *counter_p)`
Deregister given counter.

Return zero(0) or negative error code.

Parameters

- `counter_p`: Counter to deregister.

`int fs_parameter_init (struct fs_parameter_t *self_p, const FAR char * path_p, fs_paramet`
Initialize given parameter.

Return zero(0) or negative error code.

Parameters

- `self_p`: Parameter to initialize.
- `path_p`: Path to register.

- `set_cb`: Callback function set set the parameter value.
- `print_cb`: Callback function set print the parameter value.
- `value_p`: Value storage area.

int **fs_parameter_register** (struct *fs_parameter_t* **parameter_p*)

Register given parameter.

Return zero(0) or negative error code.

Parameters

- `parameter_p`: Parameter to register.

int **fs_parameter_deregister** (struct *fs_parameter_t* **parameter_p*)

Deregister given parameter.

Return zero(0) or negative error code.

Parameters

- `parameter_p`: Parameter to deregister.

int **fs_parameter_int_set** (void **value_p*, const char **src_p*)

Integer parameter setter function callback

Return zero(0) or negative error code.

Parameters

- `value_p`: Buffer the new value should be written to.
- `src_p`: Value to set as a string.

int **fs_parameter_int_print** (void **chout_p*, void **value_p*)

Integer parameter printer function callback

Return zero(0) or negative error code.

Parameters

- `chout_p`: Channel to write the formatted value to.
- `value_p`: Value to format and print to the output channel.

struct **fs_filesystem_spiffs_config_t**

#include <fs.h> A SPIFFS file system.

Public Members

struct *spiffs_config_t* ***config_p**

uint8_t ***workspace_p**

uint8_t ***buf_p**

size_t **size**

struct **fs_filesystem_spiffs_config_t::@27** fs_filesystem_spiffs_config_t::fdworkspace

```

    struct fs_filesystem_spiffs_config_t::@28    fs_filesystem_spiffs_config_t::cache
struct fs_filesystem_fat16_t
    #include <fs.h> A FAT16 file system.

```

Public Members

```

    struct fat16_t *fat16_p
struct fs_filesystem_t

```

Public Members

```

    const char *name_p
    fs_type_t type
    struct fat16_t *fat16_p
    struct spiffs_t *spiffs_p
    union fs_filesystem_t::@29    fs_filesystem_t::fs
    struct fs_filesystem_spiffs_config_t *spiffs_p
    union fs_filesystem_t::@30    fs_filesystem_t::config
    struct fs_filesystem_t *next_p
struct fs_file_t

```

Public Members

```

    struct fs_filesystem_t *filesystem_p
    struct fat16_file_t fat16
    spiffs_file_t spiffs
    union fs_file_t::@31    fs_file_t::u
struct fs_stat_t
    #include <fs.h> Path stats.

```

Public Members

```

    uint32_t size
    spiffs_obj_type_t type
struct fs_command_t

```

Public Members

```
const FAR char* fs_command_t::path_p
fs_callback_t callback
void *arg_p
struct fs_command_t *next_p
struct fs_counter_t
```

Public Members

```
struct fs_command_t command
long long unsigned int fs_counter_t::value
struct fs_counter_t *next_p
struct fs_parameter_t
```

Public Members

```
struct fs_command_t command
fs_parameter_set_callback_t set_cb
fs_parameter_print_callback_t print_cb
void *value_p
struct fs_parameter_t *next_p
struct fs_dir_t
```

Public Members

```
struct fs_filesystem_t *filesystem_p
struct fat16_dir_t fat16
struct spiffs_dir_t spiffs
union fs_dir_t::@32 fs_dir_t::u
struct fs_dir_entry_t
```

Public Members

```
char name[256]
int type
size_t size
struct date_t latest_mod_date
```

spiffs — SPI Flash File System

The source code of this module is based on <https://github.com/pellepl/spiffs>.

About

Spiffs is a file system intended for SPI NOR flash devices on embedded targets.

Spiffs is designed with following characteristics in mind:

- Small (embedded) targets, sparse RAM without heap.
- Only big areas of data (blocks) can be erased.
- An erase will reset all bits in block to ones.
- Writing pulls one to zeroes.
- Zeroes can only be pulled to ones by erase.
- Wear leveling.

Source code: [src/filesystems/spiffs.h](#), [src/filesystems/spiffs.c](#)

Test code: [tst/filesystems/spiffs/main.c](#)

Defines

```
SPIFFS_OK
SPIFFS_ERR_NOT_MOUNTED
SPIFFS_ERR_FULL
SPIFFS_ERR_NOT_FOUND
SPIFFS_ERR_END_OF_OBJECT
SPIFFS_ERR_DELETED
SPIFFS_ERR_NOT_FINALIZED
SPIFFS_ERR_NOT_INDEX
SPIFFS_ERR_OUT_OF_FILE_DESCS
SPIFFS_ERR_FILE_CLOSED
SPIFFS_ERR_FILE_DELETED
SPIFFS_ERR_BAD_DESCRIPTOR
SPIFFS_ERR_IS_INDEX
SPIFFS_ERR_IS_FREE
SPIFFS_ERR_INDEX_SPAN_MISMATCH
SPIFFS_ERR_DATA_SPAN_MISMATCH
SPIFFS_ERR_INDEX_REF_FREE
```

`SPIFFS_ERR_INDEX_REF_LU`
`SPIFFS_ERR_INDEX_REF_INVALID`
`SPIFFS_ERR_INDEX_FREE`
`SPIFFS_ERR_INDEX_LU`
`SPIFFS_ERR_INDEX_INVALID`
`SPIFFS_ERR_NOT_WRITABLE`
`SPIFFS_ERR_NOT_READABLE`
`SPIFFS_ERR_CONFLICTING_NAME`
`SPIFFS_ERR_NOT_CONFIGURED`
`SPIFFS_ERR_NOT_A_FS`
`SPIFFS_ERR_MOUNTED`
`SPIFFS_ERR_ERASE_FAIL`
`SPIFFS_ERR_MAGIC_NOT_POSSIBLE`
`SPIFFS_ERR_NO_DELETED_BLOCKS`
`SPIFFS_ERR_FILE_EXISTS`
`SPIFFS_ERR_NOT_A_FILE`
`SPIFFS_ERR_RO_NOT_IMPL`
`SPIFFS_ERR_RO_ABORTED_OPERATION`
`SPIFFS_ERR_PROBE_TOO_FEW_BLOCKS`
`SPIFFS_ERR_PROBE_NOT_A_FS`
`SPIFFS_ERR_NAME_TOO_LONG`
`SPIFFS_ERR_INTERNAL`
`SPIFFS_ERR_TEST`
`SPIFFS_DBG (...)`
`SPIFFS_GC_DBG (...)`
`SPIFFS_CACHE_DBG (...)`
`SPIFFS_CHECK_DBG (...)`
`SPIFFS_APPEND`
Any write to the filehandle is appended to end of the file.
`SPIFFS_O_APPEND`
`SPIFFS_TRUNC`
If the opened file exists, it will be truncated to zero length before opened.
`SPIFFS_O_TRUNC`
`SPIFFS_CREAT`
If the opened file does not exist, it will be created before opened.
`SPIFFS_O_CREAT`

SPIFFS_RDONLY

The opened file may only be read.

SPIFFS_O_RDONLY**SPIFFS_WRONLY**

The opened file may only be written.

SPIFFS_O_WRONLY**SPIFFS_RDWR**

The opened file may be both read and written.

SPIFFS_O_RDWR**SPIFFS_DIRECT**

Any writes to the filehandle will never be cached but flushed directly.

SPIFFS_O_DIRECT**SPIFFS_EXCL**

If SPIFFS_O_CREAT and SPIFFS_O_EXCL are set, SPIFFS_open() shall fail if the file exists.

SPIFFS_O_EXCL**SPIFFS_SEEK_SET****SPIFFS_SEEK_CUR****SPIFFS_SEEK_END****SPIFFS_TYPE_FILE****SPIFFS_TYPE_DIR****SPIFFS_TYPE_HARD_LINK****SPIFFS_TYPE_SOFT_LINK****SPIFFS_LOCK** (fs)**SPIFFS_UNLOCK** (fs)

Typedefs

```
typedef int16_t spiffs_file_t
```

Spiffs file descriptor index type. must be signed.

```
typedef uint16_t spiffs_flags_t
```

Spiffs file descriptor flags.

```
typedef uint16_t spiffs_mode_t
```

Spiffs file mode.

```
typedef uint8_t spiffs_obj_type_t
```

Object type.

```
typedef int32_t (*spiffs_read_cb_t) (uint32_t addr, uint32_t size, uint8_t *dst_p)
```

Spi read call function type.

```
typedef int32_t (*spiffs_write_cb_t) (uint32_t addr, uint32_t size, uint8_t *src_p)
```

Spi write call function type.

```
typedef int32_t (*spiffs_erase_cb_t) (uint32_t addr, uint32_t size)
```

Spi erase call function type.

```
typedef void (*spiffs_check_callback_t)(enum spiffs_check_type_t type, enum spiffs_check_report_t report, uint32_t arg1, uint32_t arg2)
```

File system check callback function.

```
typedef void (*spiffs_file_callback_t)(struct spiffs_t *fs_p, enum spiffs_fileop_type_t op, spiffs_obj_id_t obj_id, spiffs_page_ix_t pix)
```

File system listener callback function.

```
typedef spiffs_block_ix_t spiffs_block_ix
```

```
typedef spiffs_page_ix_t spiffs_page_ix
```

```
typedef spiffs_obj_id_t spiffs_obj_id
```

```
typedef spiffs_span_ix_t spiffs_span_ix
```

```
typedef struct spiffs_t spiffs
```

```
typedef spiffs_file_t spiffs_file
```

```
typedef spiffs_flags_t spiffs_flags
```

```
typedef spiffs_obj_type_t spiffs_obj_type
```

```
typedef spiffs_mode_t spiffs_mode
```

```
typedef enum spiffs_fileop_type_t spiffs_fileop_type
```

```
typedef struct spiffs_config_t spiffs_config
```

```
typedef spiffs_check_callback_t spiffs_check_callback
```

```
typedef struct spiffs_dirent_t spiffs_dirent
```

```
typedef struct spiffs_dir_t spiffs_DIR
```

```
typedef spiffs_file_callback_t spiffs_file_callback
```

Enums

```
enum spiffs_check_type_t
```

File system check callback report operation.

Values:

```
SPIFFS_CHECK_LOOKUP = 0
```

```
SPIFFS_CHECK_INDEX
```

```
SPIFFS_CHECK_PAGE
```

```
enum spiffs_check_report_t
```

File system check callback report type.

Values:

```
SPIFFS_CHECK_PROGRESS = 0
```

```
SPIFFS_CHECK_ERROR
```

```
SPIFFS_CHECK_FIX_INDEX
```

```
SPIFFS_CHECK_FIX_LOOKUP
```

```
SPIFFS_CHECK_DELETE_ORPHANED_INDEX
```

SPIFFS_CHECK_DELETE_PAGE

SPIFFS_CHECK_DELETE_BAD_FILE

enum spiffs_fileop_type_t

File system listener callback operation.

Values:

SPIFFS_CB_CREATED = 0

The file has been created.

SPIFFS_CB_UPDATED

The file has been updated or moved to another page.

SPIFFS_CB_DELETED

The file has been deleted.

Functions

int32_t spiffs_mount (**struct** *spiffs_t* *self_p, **struct** *spiffs_config_t* *config_p, uint8_t *work_p, uint8_t *fd_space_p, uint32_t fd_space_size, void *cache_p, uint32_t cache_size, *spiffs_check_callback_t* check_cb)

Initializes the file system dynamic parameters and mounts the filesystem. If SPIFFS_USE_MAGIC is enabled the mounting may fail with SPIFFS_ERR_NOT_A_FS if the flash does not contain a recognizable file system. In this case, SPIFFS_format must be called prior to remounting.

Return zero(0) or negative error code.

Parameters

- self_p: The file system struct.
- config_p: The physical and logical configuration of the file system.
- work_p: A memory work buffer comprising 2*config->log_page_size bytes used throughout all file system operations
- fd_space_p: Memory for file descriptors.
- fd_space_size: Memory size of file descriptors.
- cache_p: Memory for cache, may be NULL.
- cache_size: Memory size of cache.
- check_cb: Callback function for reporting during consistency checks.

void spiffs_unmount (**struct** *spiffs_t* *self_p)

Unmounts the file system. All file handles will be flushed of any cached writes and closed.

Return void.

Parameters

- self_p: The file system struct.

int32_t spiffs_creat (**struct** *spiffs_t* *self_p, **const** char *path_p, *spiffs_mode_t* mode)

Creates a new file.

Return zero(0) or negative error code.

Parameters

- `self_p`: The file system struct.
- `path_p`: The path of the new file.
- `mode`: Ignored, for posix compliance.

spiffs_file_t **spiffs_open** (**struct** *spiffs_t* **self_p*, **const** char **path_p*, *spiffs_flags_t* *flags*, *spiffs_mode_t* *mode*)

Opens/creates a file.

Parameters

- `self_p`: The file system struct.
- `path_p`: The path of the new file.
- `flags`: The flags for the open command, can be combinations of SPIFFS_O_APPEND, SPIFFS_O_TRUNC, SPIFFS_O_CREAT, SPIFFS_O_RDONLY, SPIFFS_O_WRONLY, SPIFFS_O_RDWR, SPIFFS_O_DIRECT, SPIFFS_O_EXCL.
- `mode`: Ignored, for posix compliance.

spiffs_file_t **spiffs_open_by_dirent** (**struct** *spiffs_t* **self_p*, **struct** *spiffs_dirent_t* **ent_p*, *spiffs_flags_t* *flags*, *spiffs_mode_t* *mode*)

Opens a file by given dir entry.

Optimization purposes, when traversing a file system with SPIFFS_readdir a normal SPIFFS_open would need to traverse the filesystem again to find the file, whilst SPIFFS_open_by_dirent already knows where the file resides.

Parameters

- `self_p`: The file system struct.
- `e_p`: The dir entry to the file.
- `flags`: The flags for the open command, can be combinations of SPIFFS_APPEND, SPIFFS_TRUNC, SPIFFS_CREAT, SPIFFS_RD_ONLY, SPIFFS_WR_ONLY, SPIFFS_RDWR, SPIFFS_DIRECT. SPIFFS_CREAT will have no effect in this case.
- `mode`: Ignored, for posix compliance.

spiffs_file_t **spiffs_open_by_page** (**struct** *spiffs_t* **self_p*, *spiffs_page_ix_t* *page_ix*, *spiffs_flags_t* *flags*, *spiffs_mode_t* *mode*)

Opens a file by given page index.

Optimization purposes, opens a file by directly pointing to the page index in the spi flash. If the page index does not point to a file header SPIFFS_ERR_NOT_A_FILE is returned.

Parameters

- `self_p`: The file system struct.
- `page_ix`: The page index.
- `flags`: The flags for the open command, can be combinations of SPIFFS_APPEND, SPIFFS_TRUNC, SPIFFS_CREAT, SPIFFS_RD_ONLY, SPIFFS_WR_ONLY, SPIFFS_RDWR, SPIFFS_DIRECT. SPIFFS_CREAT will have no effect in this case.
- `mode`: Ignored, for posix compliance.

`int32_t spiffs_read (struct spiffs_t *self_p, spiffs_file_t fh, void *buf_p, int32_t len)`
 Reads from given filehandle.

Return Number of bytes read or negative error code.

Parameters

- *self_p*: The file system struct.
- *fh*: The filehandle.
- *buf_p*: Where to put read data.
- *len*: How much to read.

`int32_t spiffs_write (struct spiffs_t *self_p, spiffs_file_t fh, void *buf_p, int32_t len)`
 Writes to given filehandle.

Return Number of bytes written, or negative error code.

Parameters

- *self_p*: The file system struct.
- *fh*: The filehandle.
- *buf_p*: The data to write.
- *len*: How much to write.

`int32_t spiffs_lseek (struct spiffs_t *self_p, spiffs_file_t fh, int32_t offs, int whence)`
 Moves the read/write file offset. Resulting offset is returned or negative if error.

`lseek(fs, fd, 0, SPIFFS_SEEK_CUR)` will thus return current offset.

If `SPIFFS_SEEK_CUR`, the file offset shall be set to its current location plus offset.

Parameters

- *self_p*: The file system struct.
- *fh*: The filehandle.
- *offs*: How much/where to move the offset.
- *whence*: If `SPIFFS_SEEK_SET`, the file offset shall be set to offset bytes.

If `SPIFFS_SEEK_END`, the file offset shall be set to the size of the file plus offse, which should be negative.

Return zero(0) or negative error code.

`int32_t spiffs_remove (struct spiffs_t *self_p, const char *path_p)`
 Removes a file by path.

Return zero(0) or negative error code.

Parameters

- *self_p*: The file system struct.
- *path_p*: The path of the file to remove.

`int32_t spiffs_fremove (struct spiffs_t *self_p, spiffs_file_t fh)`
 Removes a file by filehandle.

Return zero(0) or negative error code.

Parameters

- `self_p`: The file system struct.
- `fh`: The filehandle of the file to remove.

`int32_t spiffs_stat (struct spiffs_t *self_p, const char *path_p, struct spiffs_stat_t *stat_p)`
Gets file status by path.

Return zero(0) or negative error code.

Parameters

- `self_p`: The file system struct.
- `path_p`: The path of the file to stat.
- `stat_p`: The stat struct to populate.

`int32_t spiffs_fstat (struct spiffs_t *self_p, spiffs_file_t fh, struct spiffs_stat_t *stat_p)`
Gets file status by filehandle.

Return zero(0) or negative error code.

Parameters

- `self_p`: The file system struct.
- `fh`: The filehandle of the file to stat.
- `stat_p`: The stat struct to populate.

`int32_t spiffs_fflush (struct spiffs_t *self_p, spiffs_file_t fh)`
Flushes all pending write operations from cache for given file.

Return zero(0) or negative error code.

Parameters

- `self_p`: The file system struct.
- `fh`: The filehandle of the file to flush.

`int32_t spiffs_close (struct spiffs_t *self_p, spiffs_file_t fh)`
Closes a filehandle. If there are pending write operations, these are finalized before closing.

Return zero(0) or negative error code.

Parameters

- `self_p`: The file system struct.
- `fh`: The filehandle of the file to close.

`int32_t spiffs_rename (struct spiffs_t *self_p, const char *old_path_p, const char *new_path_p)`
Renames a file.

Return zero(0) or negative error code.

Parameters

- `self_p`: The file system struct.
- `old_path_p`: Path of file to rename.
- `new_path_p`: New path of file.

`int32_t spiffs_errno (struct spiffs_t *self_p)`

Returns last error of last file operation.

Return zero(0) or negative error code.

Parameters

- `self_p`: The file system struct.

`void spiffs_clearerr (struct spiffs_t *self_p)`

Clears last error.

Return void.

Parameters

- `self_p`: The file system struct.

`struct spiffs_dir_t *spiffs_opendir (struct spiffs_t *self_p, const char *name_p, struct spiffs_dir_t *dir_p)`

Opens a directory stream corresponding to the given name. The stream is positioned at the first entry in the directory. On hydrogen builds the name argument is ignored as hydrogen builds always correspond to a flat file structure - no directories.

Parameters

- `self_p`: The file system struct.
- `name_p`: The name of the directory.
- `dir_p`: Pointer the directory stream to be populated.

`int32_t spiffs_closedir (struct spiffs_dir_t *dir_p)`

Closes a directory stream

Return zero(0) or negative error code.

Parameters

- `dir_p`: The directory stream to close.

`struct spiffs_dirent_t *spiffs_readdir (struct spiffs_dir_t *dir_p, struct spiffs_dirent_t *ent_p)`

Reads a directory into given spifs_dirent struct.

Return NULL if error or end of stream, else given dirent is returned.

Parameters

- `dir_p`: Pointer to the directory stream.
- `ent_p`: The dirent struct to be populated.

`int32_t spiffs_check (struct spiffs_t *self_p)`

Runs a consistency check on given filesystem.

Return zero(0) or negative error code.

Parameters

- `self_p`: The file system struct.

`int32_t spiffs_info (struct spiffs_t *self_p, uint32_t *total_p, uint32_t *used_p)`

Returns number of total bytes available and number of used bytes. This is an estimation, and depends on if there are many files with little data or few files with much data.

NB: If used number of bytes exceeds total bytes, a SPIFFS_check should run. This indicates a power loss in midst of things. In worst case (repeated powerlosses in mending or gc) you might have to delete some files.

Return zero(0) or negative error code.

Parameters

- `self_p`: The file system struct.
- `total_p`: Total number of bytes in filesystem.
- `used_p`: Used number of bytes in filesystem.

`int32_t spiffs_format (struct spiffs_t *self_p)`

Formats the entire file system. All data will be lost. The filesystem must not be mounted when calling this.

NB: formatting is awkward. Due to backwards compatibility, SPIFFS_mount MUST be called prior to formatting in order to configure the filesystem. If SPIFFS_mount succeeds, SPIFFS_unmount must be called before calling SPIFFS_format. If SPIFFS_mount fails, SPIFFS_format can be called directly without calling SPIFFS_unmount first.

Return zero(0) or negative error code.

Parameters

- `self_p`: The file system struct.

`uint8_t spiffs_mounted (struct spiffs_t *self_p)`

Returns nonzero if spiffs is mounted, or zero if unmounted.

Parameters

- `self_p`: The file system struct.

`int32_t spiffs_gc_quick (struct spiffs_t *self_p, uint16_t max_free_pages)`

Tries to find a block where most or all pages are deleted, and erase that block if found. Does not care for wear levelling. Will not move pages around.

If parameter `max_free_pages` are set to 0, only blocks with only deleted pages will be selected.

NB: the garbage collector is automatically called when spiffs needs free pages. The reason for this function is to give possibility to do background tidying when user knows the system is idle.

Use with care.

Setting `max_free_pages` to anything larger than zero will eventually wear flash more as a block containing free pages can be erased.

Will set `err_no` to SPIFFS_OK if a block was found and erased, SPIFFS_ERR_NO_DELETED_BLOCK if no matching block was found, or other error.

Return zero(0) or negative error code.

Parameters

- `self_p`: The file system struct.
- `max_free_pages`: maximum number allowed free pages in block.

`int32_t spiffs_gc (struct spiffs_t *self_p, uint32_t size)`

Will try to make room for given amount of bytes in the filesystem by moving pages and erasing blocks. If it is physically impossible, `err_no` will be set to `SPIFFS_ERR_FULL`. If there already is this amount (or more) of free space, `SPIFFS_gc` will silently return. It is recommended to call `SPIFFS_info` before invoking this method in order to determine what amount of bytes to give.

NB: the garbage collector is automatically called when spiffs needs free pages. The reason for this function is to give possibility to do background tidying when user knows the system is idle.

Use with care.

Return zero(0) or negative error code.

Parameters

- `self_p`: The file system struct.
- `size`: Amount of bytes that should be freed.

`int32_t spiffs_eof (struct spiffs_t *self_p, spiffs_file_t fh)`

Check if EOF reached.

Return zero(0) or negative error code.

Parameters

- `self_p`: The file system struct.
- `fh`: The filehandle of the file to check.

`int32_t spiffs_tell (struct spiffs_t *self_p, spiffs_file_t fh)`

Get position in file.

Return zero(0) or negative error code.

Parameters

- `self_p`: The file system struct.
- `fh`: The filehandle of the file to check.

`int32_t spiffs_set_file_callback_func (struct spiffs_t *self_p, spiffs_file_callback_t cb_func)`

Registers a callback function that keeps track on operations on file headers. Do note, that this callback is called from within internal spiffs mechanisms. Any operations on the actual file system being callbacked from in this callback will mess things up for sure - do not do this. This can be used to track where files are and move around during garbage collection, which in turn can be used to build location tables in ram. Used in conjunction with `SPIFFS_open_by_page` this may improve performance when opening a lot of files. Must be invoked after mount.

Return zero(0) or negative error code.

Parameters

- `self_p`: The file system struct.
- `cb_func`: The callback on file operations.

struct spiffs_config_t
#include <spiffs.h> Spiffs spi configuration struct.

Public Members

spiffs_read_cb_t **hal_read_f**
Physical read function.

spiffs_write_cb_t **hal_write_f**
Physical write function.

spiffs_erase_cb_t **hal_erase_f**
Physical erase function.

uint32_t **phys_size**
Physical size of the spi flash.

uint32_t **phys_addr**
Physical offset in spi flash used for spiffs, must be on block boundary.

uint32_t **phys_erase_block**
Physical size when erasing a block.

uint32_t **log_block_size**
Logical size of a block, must be on physical block size boundary and must never be less than a physical block.

uint32_t **log_page_size**
Logical size of a page, must be at least log_block_size /
1.

struct spiffs_t

Public Members

struct spiffs_config_t **cfg**
File system configuration.

uint32_t **block_count**
Number of logical blocks.

spiffs_block_ix_t **free_cursor_block_ix**
Cursor for free blocks, block index.

int **free_cursor_obj_lu_entry**
Cursor for free blocks, entry index.

spiffs_block_ix_t **cursor_block_ix**
Cursor when searching, block index.

int **cursor_obj_lu_entry**
Cursor when searching, entry index.

uint8_t ***lu_work**
Primary work buffer, size of a logical page.

uint8_t ***work**
Secondary work buffer, size of a logical page.

```

uint8_t *fd_space
    File descriptor memory area.

uint32_t fd_count
    Available file descriptors.

int32_t err_code
    Last error.

uint32_t free_blocks
    Current number of free blocks.

uint32_t stats_p_allocated
    Current number of busy pages.

uint32_t stats_p_deleted
    Current number of deleted pages.

uint8_t cleaning
    Flag indicating that garbage collector is cleaning.

spiffs_obj_id_t max_erase_count
    Max erase count amongst all blocks.

spiffs_check_callback_t check_cb_f
    Check callback function.

spiffs_file_callback_t file_cb_f
    File callback function.

uint8_t mounted
    Mounted flag.

void *user_data
    User data.

uint32_t config_magic
    Config magic.

struct spiffs_stat_t
    #include <spiffs.h> Spiffs file status struct.

```

Public Members

```

spiffs_obj_id_t obj_id

uint32_t size

spiffs_obj_type_t type

spiffs_page_ix_t pix

uint8_t name[SPIFFS_OBJ_NAME_LEN]

struct spiffs_dirent_t

```

Public Members

```

spiffs_obj_id_t obj_id

uint8_t name[SPIFFS_OBJ_NAME_LEN]

```

```
spiffs_obj_type_t type
uint32_t size
spiffs_page_ix_t pix
struct spiffs_dir_t
```

Public Members

```
struct spiffs_t *fs
spiffs_block_ix_t block
int entry
```

1.6.5 inet

The inet package on [Github](#).

Modules:

`http_server` — HTTP server

Source code: [src/inet/http_server.h](#), [src/inet/http_server.c](#)

Test code: [tst/inet/http_server/main.c](#)

Test coverage: [src/inet/http_server.c](#)

Typedefs

```
typedef int (*http_server_route_callback_t)(struct http_server_connection_t *connection_p,
                                             struct http_server_request_t *request_p)
```

Enums

```
enum http_server_request_action_t
    Values:
    http_server_request_action_get_t = 0
    http_server_request_action_post_t = 1
enum http_server_content_type_t
    Content type.
    Values:
    http_server_content_type_text_plain_t = 0
    http_server_content_type_text_html_t = 1
```

enum http_server_response_code_t

Response codes.

Values:

http_server_response_code_200_ok_t = 200

http_server_response_code_401_unauthorized_t = 401

http_server_response_code_404_not_found_t = 404

enum http_server_connection_state_t

Connection state.

Values:

http_server_connection_state_free_t = 0

http_server_connection_state_allocated_t

Functions

int http_server_init (**struct** *http_server_t* *self_p, **struct** *http_server_listener_t* *listener_p, **struct** *http_server_connection_t* *connections_p, **const** char *root_path_p, **const** **struct** *http_server_route_t* *routes_p, *http_server_route_callback_t* on_no_route)

Initialize given http server with given root path and maximum number of clients.

Return zero(0) or negative error code.

Parameters

- self_p: Http server to initialize.
- listener_p: Listener.
- connections_p: A NULL terminated list of connections.
- root_path_p: Working directory for the connection threads.
- routes_p: An array of routes.
- on_no_route: Callback called for all requests without a matching route in route_p.

int http_server_start (**struct** *http_server_t* *self_p)

Start given HTTP server.

Spawn the threads and start listening for connections.

Return zero(0) or negative error code.

Parameters

- self_p: Http server.

int http_server_stop (**struct** *http_server_t* *self_p)

Stop given HTTP server.

Closes the listener and all open connections, and then kills the threads.

Return zero(0) or negative error code.

Parameters

- self_p: Http server.

```
int http_server_response_write (struct http_server_connection_t *connection_p, struct
                                http_server_request_t *request_p, struct http_server_response_t
                                *response_p)
```

Write given HTTP response to given connected client. This function should only be called from the route callbacks to respond to given request.

Return zero(0) or negative error code.

Parameters

- `connection_p`: Current connection.
- `request_p`: Current request.
- `response_p`: Current response. If `buf_p` in the response to NULL this function will only write the HTTP header, including the size, to the socket. After this function returns write the payload by calling `socket_write()`.

```
struct http_server_request_t
#include <http_server.h> HTTP request.
```

Public Members

http_server_request_action_t **action**

char **path**[64]

int **present**

char **value**[64]

struct http_server_request_t::@34::@35 http_server_request_t::sec_websocket_key

struct http_server_request_t::@34::@36 http_server_request_t::content_type

long **value**

struct http_server_request_t::@34::@37 http_server_request_t::content_length

struct http_server_request_t::@34::@38 http_server_request_t::authorization

struct http_server_request_t::@34 http_server_request_t::headers

```
struct http_server_response_t
#include <http_server.h> HTTP response.
```

Public Members

int **type**

http_server_response_code_t **code**

const char ***buf_p**

size_t **size**

struct http_server_response_t::@39 http_server_response_t::content

```
struct http_server_listener_t
```

Public Members

```

const char *address_p
int port
const char *name_p
void *buf_p
size_t size
struct http_server_listener_t::@40::@41  http_server_listener_t::stack
struct thrd_t *id_p
struct http_server_listener_t::@40  http_server_listener_t::thrd
struct socket_t socket
struct http_server_connection_t

```

Public Members

```

http_server_connection_state_t state
const char *name_p
void *buf_p
size_t size
struct http_server_connection_t::@42::@43  http_server_connection_t::stack
struct thrd_t *id_p
struct http_server_connection_t::@42  http_server_connection_t::thrd
struct http_server_t *self_p
struct socket_t socket
struct event_t events
struct http_server_route_t
#include <http_server.h> Call given callback for given path.

```

Public Members

```

const char *path_p
http_server_route_callback_t callback
struct http_server_t

```

Public Members

```

const char *root_path_p
const struct http_server_route_t *routes_p
http_server_route_callback_t on_no_route

```

```
struct http_server_listener_t *listener_p
struct http_server_connection_t *connections_p
struct event_t events
```

http_websocket_client — HTTP websocket client

Source code: `src/inet/http_websocket_client.h`, `src/inet/http_websocket_client.c`

Test code: `tst/inet/http_websocket_client/main.c`

Test coverage: `src/inet/http_websocket_client.c`

Functions

```
int http_websocket_client_init (struct http_websocket_client_t *self_p, const char *server_p, int
                                port, const char *path_p)
```

Initialize given http.

Return zero(0) or negative error code.

Parameters

- *self_p*: Http to initialize.
- *server_p*: Server hostname to connect to.
- *port*: Port to connect to.
- *path_p*: Path.

```
int http_websocket_client_connect (struct http_websocket_client_t *self_p)
```

Connect given http to the server.

Return zero(0) or negative error code.

Parameters

- *self_p*: Http to connect.

```
int http_websocket_client_disconnect (struct http_websocket_client_t *self_p)
```

Disconnect given http from the server.

Return zero(0) or negative error code.

Parameters

- *self_p*: Http to connect.

```
ssize_t http_websocket_client_read (struct http_websocket_client_t *self_p, void *buf_p, size_t
                                      size)
```

Read from given http.

Return Number of bytes read or negative error code.

Parameters

- `self_p`: Http to read from.
- `buf_p`: Buffer to read into.
- `size`: Number of bytes to read..

`ssize_t http_websocket_client_write` (`struct http_websocket_client_t *self_p`, `int type`, `const void *buf_p`, `uint32_t size`)

Write given data to given http.

Return Number of bytes written or negative error code.

Parameters

- `self_p`: Http to write to.
- `buf_p`: Buffer to write.
- `size`: Number of bytes to write.

`struct http_websocket_client_t`
`#include <http_websocket_client.h>`

Public Members

```
struct socket_t socket
const char *host_p
int port
struct http_websocket_client_t::@44 http_websocket_client_t::server
size_t left
struct http_websocket_client_t::@45 http_websocket_client_t::frame
const char *path_p
```

`http_websocket_server` — HTTP websocket server

Source code: `src/inet/http_websocket_server.h`, `src/inet/http_websocket_server.c`

Test code: `tst/inet/http_websocket_server/main.c`

Test coverage: `src/inet/http_websocket_server.c`

Functions

`int http_websocket_server_init` (`struct http_websocket_server_t *self_p`, `struct socket_t *socket_p`)
 Initialize given websocket server. The server uses the http module interface to communicate with the client.

Return zero(0) or negative error code.

Parameters

- `self_p`: Http to initialize.

- `socket_p`: Connected socket.

int **http_websocket_server_handshake** (struct *http_websocket_server_t* *self_p, struct *http_server_request_t* *request_p)

Read the handshake request from the client and send the handshake response.

Return zero(0) or negative error code.

Parameters

- `self_p`: Websocket server.
- `request_p`: Read handshake request.

ssize_t **http_websocket_server_read** (struct *http_websocket_server_t* *self_p, int *type_p, void *buf_p, size_t size)

Read a message from given websocket.

Return Number of bytes read or negative error code.

Parameters

- `self_p`: Websocket to read from.
- `type_p`: Read message type.
- `buf_p`: Buffer to read into.
- `size`: Number of bytes to read. Longer messages will be truncated and the leftover data dropped.

ssize_t **http_websocket_server_write** (struct *http_websocket_server_t* *self_p, int type, const void *buf_p, uint32_t size)

Write given message to given websocket.

Return Number of bytes written or negative error code.

Parameters

- `self_p`: Websocket to write to.
- `type`: One of `HTTP_TYPE_TEXT` and `HTTP_TYPE_BINARY`.
- `buf_p`: Buffer to write.
- `size`: Number of bytes to write.

struct **http_websocket_server_t**
#include <http_websocket_server.h>

Public Members

struct *socket_t* ***socket_p**

inet — Internet utilities

Source code: [src/inet/inet.h](#), [src/inet/inet.c](#)

Test code: [tst/inet/inet/inet.c](#)

Test coverage: [src/inet/inet.c](#)

Functions

int **inet_module_init** (void)

Initialize the inet module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

int **inet_aton** (const char *src_p, struct *inet_ip_addr_t* *dst_p)

Convert the Internet host address src_p from the IPv4 numbers-and-dots notation into binary form (in network byte order) and stores it in the structure that dst_p points to.

The address supplied in src_p can have one of the following forms:

- a.b.c.d Each of the four numeric parts specifies a byte of the address; the bytes are assigned in left-to-right order to produce the binary address.

Return zero(0) or negative error code.

Parameters

- src_p: Address a.b.c.d to convert into a number.
- dst_p: Converted address.

char ***inet_ntoa** (const struct *inet_ip_addr_t* *src_p, char *dst_p)

Convert the Internet host src_p from the IPv4 binary form (in network byte order) to numbers-and-dots notation and stores it in the structure that dst_p points to.

Return Converted address pointer or NULL on failure.

Parameters

- src_p: Address to convert into a string.
- dst_p: Converted address as a string.

uint16_t **inet_checksum** (void *buf_p, size_t size)

Calculate the internet checksum of given buffer.

Return Calculated checksum.

Parameters

- buf_p: Buffer to calculate the checksum of.
- size: Size of the buffer.

struct **inet_ip_addr_t**

#include <inet.h>

Public Members

uint32_t **number**
IPv4 address.

struct inet_addr_t

Public Members

struct inet_ip_addr_t ip
IPv4 address.

uint16_t **port**
Port.

struct inet_if_ip_info_t
#include <inet.h> Interface IP information.

Public Members

struct inet_ip_addr_t address

struct inet_ip_addr_t netmask

struct inet_ip_addr_t gateway

mqtt_client — MQTT client

Source code: [src/inet/mqtt_client.h](#), [src/inet/mqtt_client.c](#)

Test code: [tst/inet/mqtt_client/main.c](#)

Test coverage: [src/inet/mqtt_client.c](#)

Typedefs

typedef size_t (***mqtt_on_publish_t**) (**struct** *mqtt_client_t* *client_p, **const** char *topic_p, void *chin_p, size_t size)
Prototype of the on-publish callback function.

Return Number of bytes read from the input channel.

Parameters

- client_p: The client.
- topic_p: The received topic.
- chin_p: The channel to read the value from.
- size: Number of bytes of the value to read from chin_p.

typedef int (***mqtt_on_error_t**) (**struct** *mqtt_client_t* *client_p, int error)
Prototype of the on-error callback function.

Return zero(0) or negative error code.

Parameters

- `client_p`: The client.
- `error`: The number of error that occurred.

Enums

`enum mqtt_client_state_t`

Values:

`mqtt_client_state_disconnected_t`

`mqtt_client_state_connected_t`

`mqtt_client_state_connecting_t`

`enum mqtt_qos_t`

Quality of Service.

Values:

`mqtt_qos_0_t = 0`

`mqtt_qos_1_t = 1`

`mqtt_qos_2_t = 2`

Functions

`int mqtt_client_init(struct mqtt_client_t *self_p, const char *name_p, struct log_object_t *log_object_p, void *chout_p, void *chin_p, mqtt_on_publish_t on_publish, mqtt_on_error_t on_error)`

Initialize given MQTT client.

Return zero(0) or negative error code.

Parameters

- `self_p`: MQTT client.
- `name_p`: Name of the thread.
- `log_object_p`: Log object.
- `chout_p`: Output channel for client to server packets.
- `chin_p`: Input channel for server to client packets.
- `on_publish`: On-publish callback function. Called when the server publishes a message.
- `on_error`: On-error callback function. Called when an error occurs.

`void *mqtt_client_main(void *arg_p)`
MQTT client thread.

Return Never returns.

Parameters

- `arg_p`: MQTT client.

int **mqtt_client_connect** (struct *mqtt_client_t* *self_p)

Establish a connection to the server.

Return zero(0) or negative error code.

Parameters

- self_p: MQTT client.

int **mqtt_client_disconnect** (struct *mqtt_client_t* *self_p)

Disconnect from the server.

Return zero(0) or negative error code.

Parameters

- self_p: MQTT client.

int **mqtt_client_ping** (struct *mqtt_client_t* *self_p)

Send a ping request to the server (broker) and wait for the ping response.

Return zero(0) or negative error code.

Parameters

- self_p: MQTT client.

int **mqtt_client_publish** (struct *mqtt_client_t* *self_p, struct *mqtt_application_message_t* *message_p)

Publish given topic.

Return zero(0) or negative error code.

Parameters

- self_p: MQTT client.
- topic_p: Topic.
- payload_p: Payload to publish. May be NULL.
- payload_size: Number of bytes in the payload.

int **mqtt_client_subscribe** (struct *mqtt_client_t* *self_p, struct *mqtt_application_message_t* *message_p)

Subscribe to given message.

Return zero(0) or negative error code.

Parameters

- self_p: MQTT client.
- message_p: The message to subscribe to. The payload part of the message is not used. The topic may use wildcards, given that the server supports it.

int **mqtt_client_unsubscribe** (struct *mqtt_client_t* *self_p, struct *mqtt_application_message_t* *message_p)

Unsubscribe from given message.

Return zero(0) or negative error code.

Parameters

- `self_p`: MQTT client.
- `message_p`: The message to unsubscribe from. Only the topic in the message is used.

```
struct mqtt_client_t
#include <mqtt_client.h> MQTT client.
```

Public Members

```
const char *name_p
struct log_object_t *log_object_p
int state
int type
void *data_p
struct mqtt_client_t::@46  mqtt_client_t::message
void *out_p
void *in_p
struct mqtt_client_t::@47  mqtt_client_t::transport
struct queue_t out
struct queue_t in
struct mqtt_client_t::@48  mqtt_client_t::control
mqtt_on_publish_t on_publish
mqtt_on_error_t on_error
struct mqtt_application_message_t
#include <mqtt_client.h> MQTT application message.
```

Public Members

```
const char *buf_p
size_t size
struct mqtt_application_message_t::@49  mqtt_application_message_t::topic
const void *buf_p
struct mqtt_application_message_t::@50  mqtt_application_message_t::payload
mqtt_qos_t qos
```

network_interface — Network interface

The network interface module has a list of all network interfaces and their states.

Network interface modules:

network_interface_slip — Serial Link Internet Protocol

Serial Line Internet Protocol (SLIP) is a link layer internet protocol used to transfer TCP/IP packets over a point-to-point serial line.

It is documented in RFC 1055.

Source code: [src/inet/network_interface/slip.h](#)

Example code: [examples/inet/slip/main.c](#)

Defines

NETWORK_INTERFACE_SLIP_FRAME_SIZE_MAX

Enums

enum network_interface_slip_state_t

Values:

NETWORK_INTERFACE_SLIP_STATE_NORMAL = 0

NETWORK_INTERFACE_SLIP_STATE_ESCAPE

Functions

int network_interface_slip_module_init (void)

Initialize the slip module.

Return zero(0) or negative error code.

int network_interface_slip_init (**struct network_interface_slip_t** *self_p, **struct inet_ip_addr_t** *ipaddr_p, **struct inet_ip_addr_t** *netmask_p, **struct inet_ip_addr_t** *gateway_p, void *chout_p)

Initialize given slip network interface with given configuration and output channel.

Return zero(0) or negative error code.

Parameters

- self_p: Slip to initialize.
- ipaddr_p: Network interface IP address.
- netmask_p: Network interface netmask.
- gateway_p: Network interface gateway.
- chout_p: Output channel.

int network_interface_slip_input (**struct network_interface_slip_t** *self_p, uint8_t data)

Input a byte into the SLIP IP stack. Normally a user thread reads one byte at a time from the UART and calls this functions with the read byte as argument.

Return Number of bytes written to the input frame or negative error code.

Parameters

- `self_p`: Slip to initialize.
- `data`: Byte to input into the stack.

struct network_interface_slip_t

Public Members

network_interface_slip_state_t **state**

struct pbuf ***pbuf_p**

uint8_t ***buf_p**

size_t **size**

struct network_interface_slip_t::@52 network_interface_slip_t::frame

void ***chout_p**

struct *network_interface_t* **network_interface**

network_interface_wifi — WiFi network interface

WiFi network interface driver modules:

network_interface_driver_esp — ESP WiFi network interface driver

Source code: `src/inet/network_interface/driver/esp.h`, `src/inet/network_interface/driver/esp.c`

Test code: `tst/inet/network_interface/wifi_esp/main.c`

Variables

struct *network_interface_wifi_driver_t* **network_interface_wifi_driver_esp_station**

struct *network_interface_wifi_driver_t* **network_interface_wifi_driver_esp_softap**

Espressif WiFi SoftAP driver callbacks. To be used as driver in the wifi network interface.

Source code: `src/inet/network_interface/wifi.h`, `src/inet/network_interface/wifi.c`

Test code: `tst/inet/network_interface/wifi_esp/main.c`

Functions

int **network_interface_wifi_module_init** (void)

Initialize the WiFi network interface module.

Return zero(0) or negative error code.

int **network_interface_wifi_init** (struct *network_interface_wifi_t* *self_p, const char *name_p,
struct *network_interface_wifi_driver_t* *driver_p, void *arg_p,
const char *ssid_p, const char *password_p)

Initialize given WiFi network interface with given configuration.

Return zero(0) or negative error code.

Parameters

- self_p: The WiFi network interface to initialize.
- name_p: Name to assign the to interface.
- driver_p: Driver virtualization callbacks to use.
- arg_p: Argument passed to the driver callbacks.
- ssid_p: Access Point SSID.
- password_p: Access Point password.

int **network_interface_wifi_start** (struct *network_interface_wifi_t* *self_p)

Start given WiFi network interface.

Return zero(0) or negative error code.

Parameters

- self_p: WiFi network interface to start.

int **network_interface_wifi_stop** (struct *network_interface_wifi_t* *self_p)

Stop given WiFi network interface.

Return zero(0) or negative error code.

Parameters

- self_p: WiFi network interface to stop.

int **network_interface_wifi_is_up** (struct *network_interface_wifi_t* *self_p)

Get the connection status of given network interface.

Return true(1) if the network interface is up, false(0) if it is down, and otherwise negative error code.

Parameters

- self_p: Network interface to get the connection status of.

int **network_interface_wifi_set_ip_info** (struct *network_interface_wifi_t* *self_p, const struct
inet_if_ip_info_t *info_p)

Set the ip address, netmask and gateway of given network interface.

Return zero(0) if the interface has valid IP information, otherwise negative error code.

Parameters

- `self_p`: Network interface.
- `info_p`: Interface IP information to set.

```
int network_interface_wifi_get_ip_info(struct network_interface_wifi_t *self_p, struct
                                     inet_if_ip_info_t *info_p)
```

Get the ip address, netmask and gateway of given network interface.

Return zero(0) if the interface has valid IP information, otherwise negative error code.

Parameters

- `self_p`: Network interface.
- `info_p`: Interface IP information. Only valid if this function returns zero(0).

```
struct network_interface_wifi_t
#include <wifi.h>
```

Public Members

```
struct network_interface_t network_interface
struct network_interface_wifi_driver_t *driver_p
void *arg_p
const char *ssid_p
const char *password_p
const struct inet_if_ip_info_t *info_p
```

```
struct network_interface_wifi_driver_t
#include <wifi.h> Driver virtualization callbacks. See the driver/ subfolder for available drivers.
```

Public Members

```
int (*init)(void *arg_p)
int (*start)(void *arg_p, const char *ssid_p, const char *password_p, const struct inet_if_ip_info_t
             *info_p)
int (*stop)(void *arg_p)
int (*is_up)(void *arg_p)
int (*set_ip_info)(void *arg_p, const struct inet_if_ip_info_t *info_p)
int (*get_ip_info)(void *arg_p, struct inet_if_ip_info_t *info_p)
```

Debug file system commands

One debug file system command is available, located in the directory `inet/network_interface/`.

Command	Description
<code>list</code>	Print a list of all registered network interfaces.

Example output from the shell:

\$ inet/network_interface/list				
NAME	STATE	ADDRESS	TX BYTES	RX BYTES
esp-wlan-ap	up	192.168.4.1	–	–
esp-wlan-sta	up	192.168.0.5	–	–

Source code: [src/inet/network_interface.h](#), [src/inet/network_interface.c](#)

Test coverage: [src/inet/network_interface.c](#)

Typedefs

typedef int (***network_interface_start_t**) (struct *network_interface_t* *netif_p)

typedef int (***network_interface_stop_t**) (struct *network_interface_t* *netif_p)

typedef int (***network_interface_is_up_t**) (struct *network_interface_t* *netif_p)

typedef int (***network_interface_set_ip_info_t**) (struct *network_interface_t* *netif_p, const struct *inet_if_ip_info_t* *info_p)

typedef int (***network_interface_get_ip_info_t**) (struct *network_interface_t* *netif_p, struct *inet_if_ip_info_t* *info_p)

Functions

int **network_interface_module_init** (void)

Initialize the network interface module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

int **network_interface_add** (struct *network_interface_t* *netif_p)

Add given network interface to the global list of network interfaces. Call `network_interface_start()` to enable the interface.

Return zero(0) or negative error code.

Parameters

- `netif_p`: Network interface to register.

int **network_interface_start** (struct *network_interface_t* *netif_p)

Start given network interface. Enables the interface in the IP stack to allow packets to be sent and received. If the interface is a WiFi station interface it will try initiate the connection to its configured access point. Use `network_interface_is_up()` to check if the interface is connected to its access point.

Return zero(0) or negative error code.

Parameters

- `netif_p`: Network interface to start.

int **network_interface_stop** (struct *network_interface_t* **netif_p*)

Stop given network interface. Disconnects from any WiFi access points and disables the interface in the IP stack. No packets can be sent or received on this interface after this function is called.

Return zero(0) or negative error code.

Parameters

- `netif_p`: Network interface to stop.

int **network_interface_is_up** (struct *network_interface_t* **netif_p*)

Get the connection status of given network interface. Packets can only be sent and received when the interface is up.

Return true(1) if the network interface is up, false(0) if it is down, and otherwise negative error code.

Parameters

- `netif_p`: Network interface to get the connection status of.

struct *network_interface_t* ***network_interface_get_by_name** (const char **name_p*)

Search the global list of network interfaces for an interface with given name and return it.

Return Found network interface or NULL if it was not found.

Parameters

- `name_p`: Name of the network interface to find.

int **network_interface_set_ip_info** (struct *network_interface_t* **netif_p*, const struct *inet_if_ip_info_t* **info_p*)

Set the IP information of given network interface.

Return zero(0) or negative error code.

Parameters

- `netif_p`: Network interface to get the IP information of.
- `info_p`: IP information to set.

int **network_interface_get_ip_info** (struct *network_interface_t* **netif_p*, struct *inet_if_ip_info_t* **info_p*)

Get the IP information of given network interface.

Return zero(0) or negative error code.

Parameters

- `netif_p`: Network interface to get the IP information of.
- `info_p`: Read IP information.

struct **network_interface_t**

Public Members

```
const char *name_p
struct inet_if_ip_info_t info
network_interface_start_t start
network_interface_stop_t stop
network_interface_is_up_t is_up
network_interface_set_ip_info_t set_ip_info
network_interface_get_ip_info_t get_ip_info
void *netif_p
struct network_interface_t *next_p
```

ping — Ping

Debug file system commands

One debug file system command is available, located in the directory `inet/ping/`.

Command	Description
<code>ping <remote host></code>	Ping a remote host by given ip address.

Example output from the shell:

```
$ inet/ping/ping 192.168.1.100
Successfully pinged '192.168.1.100' in 10 ms.
$
```

Source code: [src/inet/ping.h](#), [src/inet/ping.c](#)

Test code: [tst/inet/ping/main.c](#)

Test coverage: [src/inet/ping.c](#)

Functions

int **ping_module_init** (void)

int **ping_host_by_ip_address** (struct *inet_ip_addr_t* *address_p, struct *time_t* *timeout_p, struct *time_t* *round_trip_time_p)

Ping host by given ip address. Send an echo request packet to given host and wait for the echo reply packet. No extra payload data is transmitted, only the ICMP header.

Return zero(0) or negative error code.

Parameters

- address_p: IP address of the host to ping.
- timeout_p: Number of seconds to wait for the echo reply packet.

- `round_trip_time_p`: The time it took from sending the echo request packet to receiving the echo reply packet. Only valid if this functions returns zero(0).

socket — Internet communication

Sockets are used to communicate over IP networks. TCP and UDP are the most common transport protocols.

No more than one thread may read from a socket at any given moment. The same applies when writing to a socket. The reader and writer may be different threads, though. The behaviour is undefined if more threads use the same socket simultaneously. The application will likely crash. Add a semaphore to protect the socket if more threads need access to a socket.

Below is a TCP client example that connects to a server and sends data.

```
uint8_t buf[16];
struct socket_t tcp;
struct inet_addr_t local_addr, remote_addr;

/* Set the local and remote addresses. */
inet_aton("192.168.1.103", &local_addr.ip);
local_addr.port = 6000;
inet_aton("192.168.1.106", &remote_addr.ip);
remote_addr.port = 5000;

/* Initialize the socket and connect to the server. */
socket_open_tcp(&tcp);
socket_bind(&tcp, &local_addr);
socket_connect(&tcp, &remote_addr);

/* Send the data. */
memset(buf, 0, sizeof(buf));
socket_write(&tcp, buf, sizeof(buf));

/* Close the connection. */
socket_close(&tcp);
```

And below is the same scenario for UDP.

```
uint8_t buf[16];
struct socket_t udp;
struct socket_addr_t local_addr, remote_addr;

/* Set the local and remote addresses. */
inet_aton("192.168.1.103", &local_addr.ip);
local_addr.port = 6000;
inet_aton("192.168.1.106", &remote_addr.ip);
remote_addr.port = 5000;

/* Initialize the socket and connect to the server. */
socket_open_udp(&udp);
socket_bind(&udp, &local_addr);
socket_connect(&udp, &remote_addr);

/* Send the data. */
memset(buf, 0, sizeof(buf));
socket_send(&udp, buf, sizeof(buf));
```

```
/* Close the connection. */  
socket_close(&udp);
```

Source code: [src/inet/socket.h](#), [src/inet/socket.c](#)

Defines

SOCKET_DOMAIN_INET

SOCKET_TYPE_STREAM

TCP socket type.

SOCKET_TYPE_DGRAM

UDP socket type.

SOCKET_TYPE_RAW

RAW socket type.

SOCKET_PROTO_ICMP

Functions

int **socket_module_init** (void)

Initialize the socket module. This function will start the lwIP TCP/IP stack. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

int **socket_open_tcp** (struct *socket_t* *self_p)

Initialize given TCP socket.

Return zero(0) or negative error code.

Parameters

- self_p: Socket to initialize.

int **socket_open_udp** (struct *socket_t* *self_p)

Initialize given UDP socket.

Return zero(0) or negative error code.

Parameters

- self_p: Socket to initialize.

int **socket_open_raw** (struct *socket_t* *self_p)

Initialize given RAW socket.

Return zero(0) or negative error code.

Parameters

- `self_p`: Socket to initialize.

int **socket_open** (**struct** *socket_t* **self_p*, int *domain*, int *type*, int *protocol*)
 Initialize given socket.

Return zero(0) or negative error code.

Parameters

- `self_p`: Socket to initialize.
- `domain`: Socket domain.
- `type`: Socket type.
- `protocol`: Socket protocol.

int **socket_close** (**struct** *socket_t* **self_p*)
 Close given socket. No data transfers are allowed on after the socket has been closed.

Return zero(0) or negative error code.

Parameters

- `self_p`: Socket to close.

int **socket_bind** (**struct** *socket_t* **self_p*, **const struct** *inet_addr_t* **local_addr_p*)
 Bind given local address to given socket.

Return zero(0) or negative error code.

Parameters

- `self_p`: Socket.
- `local_addr_p`: Local address.

int **socket_listen** (**struct** *socket_t* **self_p*, int *backlog*)
 Listen for connections from remote clients. Only applicable for TCP sockets.

Return zero(0) or negative error code.

Parameters

- `self_p`: Socket to listen on.
- `backlog`: Unused.

int **socket_connect** (**struct** *socket_t* **self_p*, **const struct** *inet_addr_t* **remote_addr_p*)
 Connect to given remote address. Connecting a UDP socket sets the default remote address for outgoing datagrams. For TCP a three-way handshake with the remote peer is initiated.

Return zero(0) or negative error code.

Parameters

- `self_p`: Socket.
- `remote_addr_p`: Remote address.

int **socket_connect_by_hostname** (struct *socket_t* *self_p, const char *hostname_p, uint16_t port)

Connect to the remote device with given hostname.

In computer networking, a hostname (archaically nodename) is a label that is assigned to a device connected to a computer network and that is used to identify the device in various forms of electronic communication, such as the World Wide Web.

Return zero(0) or negative error code.

Parameters

- self_p: Socket.
- hostname_p: The hostname of the remote device to connect to.
- port: Remote device port to connect to.

int **socket_accept** (struct *socket_t* *self_p, struct *socket_t* *accepted_p, struct *inet_addr_t* *remote_addr_p)

Accept a client connect attempt. Only applicable for TCP sockets that are listening for connections.

Return zero(0) or negative error code.

Parameters

- self_p: TCP socket.
- accepted_p: New client socket of the accepted client.
- remote_addr_p: Address of the client.

ssize_t **socket_sendto** (struct *socket_t* *self_p, const void *buf_p, size_t size, int flags, const struct *inet_addr_t* *remote_addr_p)

Write data to given socket. Only used by UDP sockets.

Return Number of sent bytes or negative error code.

Parameters

- self_p: Socket to send data on.
- buf_p: Buffer to send.
- size: Size of buffer to send.
- flags: Unused.
- remote_addr_p: Remote address to send the data to.

ssize_t **socket_recvfrom** (struct *socket_t* *self_p, void *buf_p, size_t size, int flags, struct *inet_addr_t* *remote_addr_p)

Read data from given socket. Only used by UDP sockets.

Return Number of received bytes or negative error code.

Parameters

- self_p: Socket to receive data on.
- buf_p: Buffer to read into.
- size: Size of buffer to read.
- flags: Unused.

- `remote_addr_p`: Remote address to receive data from.

`ssize_t socket_write(struct socket_t *self_p, const void *buf_p, size_t size)`

Write data to given TCP or UDP socket. For UDP sockets, `socket_connect()` must have been called prior to calling this function.

Return Number of written bytes or negative error code.

Parameters

- `self_p`: Socket.
- `buf_p`: Buffer to send.
- `size`: Numer of bytes to send.

`ssize_t socket_read(struct socket_t *self_p, void *buf_p, size_t size)`

Read data from given socket.

Return Number of read bytes or negative error code.

Parameters

- `self_p`: Socket.
- `buf_p`: Buffer to read into.
- `size`: Number of bytes to read.

`ssize_t socket_size(struct socket_t *self_p)`

Get the number of input bytes currently stored in the socket. May return less bytes than number of bytes stored in the channel.

Return Number of input bytes in the socket.

Parameters

- `self_p`: Socket.

`struct socket_t`

Public Members

`struct chan_t base`

`int type`

`ssize_t left`

`struct socket_t::@53::@55::@57 socket_t::common`

`struct pbuf *pbuf_p`

`struct inet_addr_t remote_addr`

`int closed`

`struct socket_t::@53::@55::@58 socket_t::recvfrom`

`struct tcp_pcb *pcb_p`

`struct socket_t::@53::@55::@59 socket_t::accept`

```
union socket_t::@53::@55  socket_t::u
int state
void *args_p
struct thrd_t *thrd_p
struct socket_t::@53::@56  socket_t::cb
struct socket_t::@53  socket_t::input
struct socket_t::@54::@60  socket_t::cb
struct socket_t::@54  socket_t::output
void *pcb_p
```

1.6.6 oam

Operations and maintenance of an application is essential to configure, debug and monitor its operation.

The oam package on [Github](#).

console — System console

The system console is the default communication channel to an application. The console input and output channels are often terminated by a shell to enable the user to control and debug the application.

Configure the console by changing the *configuration variables* called `CONFIG_START_CONSOLE*`.

Source code: [src/oam/console.h](#), [src/oam/console.c](#)

Test coverage: [src/oam/console.c](#)

Functions

int **console_module_init** (void)

int **console_init** (void)

Initialize the console.

Return zero(0) or negative error code.

int **console_start** (void)

Start the console.

Return zero(0) or negative error code.

int **console_stop** (void)

Stop the console.

Return zero(0) or negative error code.

int **console_set_input_channel** (void *chan_p)
Set the pointer to the input channel.

Return zero(0) or negative error code.

void ***console_get_input_channel** (void)
Get the pointer to the input channel.

Return Input channel or NULL.

void ***console_set_output_channel** (void *chan_p)
Set the pointer to the output channel.

Return zero(0) or negative error code.

void ***console_get_output_channel** (void)
Get the pointer to the output channel.

Return Output channel or NULL.

service — Services

A service is as a background task. A service is either running or stopped.

Debug file system commands

Three debug file system commands is available, all located in the directory `oam/service/`.

Command	Description
<code>list</code>	List all registered services.
<code>start <service></code>	Start given service.
<code>stop <service></code>	Stop given service.

Example output from the shell:

```
$ oam/service/list
NAME                STATUS
http_server         running
ftp_server           stopped
network_manager     running
$ oam/service/start ftp_server
$ oam/service/stop http_server
$ oam/service/list
NAME                STATE
http_server         stopped
ftp_server           running
network_manager     running
```

Source code: [src/oam/service.h](#), [src/oam/service.c](#)

Test code: [tst/oam/service/main.c](#)

Test coverage: [src/oam/service.c](#)

Defines

SERVICE_CONTROL_EVENT_START

SERVICE_CONTROL_EVENT_STOP

Service stop event.

Typedefs

```
typedef enum service_status_t (*service_get_status_cb_t) (struct service_t *self_p)
```

Enums

```
enum service_status_t
```

Values:

```
    service_status_running_t = 0
```

```
    service_status_stopped_t = 1
```

Functions

```
int service_module_init (void)
```

Initialize the service module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

```
int service_init (struct service_t *self_p, const char *name_p, service_get_status_cb_t status_cb)
```

Initialize a service with given name and status callback.

Return zero(0) or negative error code.

Parameters

- *self_p*: Service to initialize.
- *name_p*: Name of the service.
- *status_callback*: Callback function returning the service status.

```
int service_start (struct service_t *self_p)
```

Start given service.

The event `SERVICE_CONTROL_EVENT_START` will be written to the control channel of given service and it's up to the service to act on this event. All services should act on all control events.

Return zero(0) or negative error code.

Parameters

- *self_p*: Service to start.

int **service_stop** (struct *service_t* **self_p*)

Stop given service.

The event `SERVICE_CONTROL_EVENT_STOP` will be written to the control channel of given service and it's up to the service to act on this event. All services should act on all control events.

Return zero(0) or negative error code.

Parameters

- *self_p*: Service to stop.

int **service_register** (struct *service_t* **service_p*)

Register given service to the global list of services.

Return zero(0) or negative error code.

Parameters

- *service_p*: Service to register.

int **service_deregister** (struct *service_t* **service_p*)

Deregister given service from the global list of services.

Return zero(0) or negative error code.

Parameters

- *service_p*: Service to deregister.

struct **service_t**

#include <service.h> A service with name and control event channel.

Public Members

const char ***name_p**

struct *event_t* **control**

service_get_status_cb_t **status_cb**

struct *service_t* ***next_p**

settings — Persistent application settings

Settings are stored in a non-volatile memory (NVM). In other words, settings are perserved after a board reset or power cycle.

Application settings are defined in an ini-file that is used to generate the c source code. A setting has a type, a size, an address and a default value, all defined in the ini-file.

Supported types are:

- `int8_t` An 8 bits signed integer.
- `int16_t` A 16 bits signed integer.
- `int32_t` A 32 bits signed integer.
- `string` An ASCII string.

The size is the number of bytes of the value. For the standard integer types the size must be the value returned by `sizeof()`. For strings it is the length of the string, including null termination.

The address for each setting is defined by the user, starting at address 0 and increasing from there.

The build system variable `SETTINGS_INI` contains the path to the ini-file used by the build system. Set this variable to the path of your application ini-file and run `make settings-generate` to generate four files; `settings.h`, `settings.c`, `settings.little-endian.bin` and `settings.big-endian.bin`.

Also add this to the Makefile: `SRC += settings.c` and include `settings.h` in the source files that access the settings.

Debug file system commands

Four debug file system commands are available, all located in the directory `oam/settings/`.

Command	Description
<code>list</code>	Print a list of the current settings.
<code>reset</code>	Overwrite the current settings values with their default values (the values defined in the ini-file values).
<code>read <name></code>	Read the value of setting <code><name></code> .
<code>write <name> <value></code>	Write <code><value></code> to setting <code><name></code> .

Example output from the shell:

```
$ oam/settings/list
NAME          TYPE      SIZE  VALUE
version       int8_t    1     1
value_1       int16_t   2    24567
value_2       int32_t   4    -57
value_3       string    16    foobar
$ oam/settings/read value_1
24567
$ oam/settings/write value_1 -5
$ oam/settings/read value_1
-5
$ oam/settings/reset
$ oam/settings/list
NAME          TYPE      SIZE  VALUE
version       int8_t    1     1
value_1       int16_t   2    24567
value_2       int32_t   4    -57
value_3       string    16    foobar
```

Example

In this example the ini-file has one setting defined, `foo`. The type is `int8_t`, the address is `0x00`, the size is 1 and the default value is `-4`.

```
[types]
foo = int8_t

[addresses]
foo = 0x00
```



```
[sizes]
foo = 1

[values]
foo = -4
```

The settings can be read and written with the functions *settings_read()* and *settings_write()*. Give the generated defines `SETTING_FOO_ADDR` and `SETTING_FOO_SIZE` as arguments to those functions.

```
int my_read_write_foo()
{
    int8_t foo;

    /* Read the foo setting. */
    if (settings_read(&foo,
                     SETTING_FOO_ADDR,
                     SETTING_FOO_SIZE) != 0) {
        return (-1);
    }

    foo -= 1;

    /* Write the foo setting. */
    if (settings_write(SETTING_FOO_ADDR,
                      &foo,
                      SETTING_FOO_SIZE) != 0) {
        return (-1);
    }

    return (0);
}
```

Source code: [src/oam/settings.h](#), [src/oam/settings.c](#)

Test code: [tst/oam/settings/main.c](#)

Test coverage: [src/oam/settings.c](#)

Defines

SETTINGS_AREA_CRC_OFFSET

Enums

enum setting_type_t

Settings types. Each setting must have be one of these types.

Values:

setting_type_int8_t = 0

setting_type_int16_t

setting_type_int32_t

setting_type_string_t

Functions

int settings_module_init (void)

Initialize the settings module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

ssize_t settings_read (void **dst_p*, size_t *src*, size_t *size*)

Read the value of given setting by address.

Return Number of words read or negative error code.

Parameters

- *dst_p*: The read value.
- *src*: Setting source address.
- *size*: Number of words to read.

ssize_t settings_write (size_t *dst*, const void **src_p*, size_t *size*)

Write given value to given setting by address.

Return Number of words written or negative error code.

Parameters

- *dst*: Destination setting address.
- *src_p*: Value to write.
- *size*: Number of bytes to write.

ssize_t settings_read_by_name (const char **name_p*, void **dst_p*, size_t *size*)

Read the value of given setting by name.

Return Number of words read or negative error code.

Parameters

- *name_p*: Setting name.
- *dst_p*: The read value.
- *size*: Size of the destination buffer.

ssize_t settings_write_by_name (const char **name_p*, const void **src_p*, size_t *size*)

Write given value to given setting by name.

Return Number of words read or negative error code.

Parameters

- *name_p*: Setting name.
- *src_p*: Value to write.

- `size`: Number of bytes to write.

int **settings_reset** (void)

Overwrite all settings with their default values.

Return zero(0) or negative error code.

struct **setting_t**

Public Members

FAR const char* **setting_t::name_p**

setting_type_t **type**

uint32_t **address**

size_t **size**

shell — Debug shell

The shell is a command line interface where the user can execute various commands to control, debug and monitor its

```
username: erik
password: *****
$
$ kernel/thrd/list
      NAME      PARENT    STATE  PRI0   CPU  LOGMASK
      main              current   0    0%    0x3f
      idle       main     ready   127   0%    0x3f
      monitor    main     ready   -80   0%    0x3f
$ history
1: kernel/thrd/list
2: history
$ logout
```

application. The shell module has a few configuration variables that can be used to tailor the shell to the application requirements. Most noticeably is the configuration variable `CONFIG_SHELL_MINIMAL`. If set to 0 all the shell functionality is built; including tab completion, cursor movement, line editing and command history. If set to 1 only the minimal functionality is built; only including tab completion and line editing at the end of the line.

See [Configuration](#) for a list of all configuration variables.

Source code: [src/oam/shell.h](#), [src/oam/shell.c](#)

Test code: [tst/oam/shell/main.c](#)

Test coverage: [src/oam/shell.c](#)

Example code: [examples/shell/main.c](#)

Functions

int **shell_module_init** (void)

Initialize the shell module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

int **shell_init** (struct *shell_t* *self_p, void *chin_p, void *chout_p, void *arg_p, const char *name_p, const char *username_p, const char *password_p)

Initialize a shell with given parameters.

Parameters

- chin_p: The shell input channel. The shell waits for commands on this channel.
- chout_p: The shell output channel. The shell writes responses on this channel.
- arg_p: User supplied argument passed to all commands.
- name_p: The shell thread name.
- username_p: Shell login username, or NULL if no username is required to use the shell.
- password_p: Shell login password. This field is unused if username_p is NULL.

void ***shell_main** (void *arg_p)

The shell main function that listens for commands on the input channel and send response on the output channel. All received commands are passed to the debug file system function `fs_call()` for execution.

Here is an example of using the shell to list and execute debug file system commands.

```
$ <tab>
drivers/
kernel/
$ kernel/ <tab>
fs/
sys/
thrd/
$ kernel/thrd/list
      NAME      STATE  PRIO   CPU  LOGMASK
      main      current    0    0%    0x0f
      idle      ready   127    0%    0x0f
      monitor    ready   -80    0%    0x0f
$
```

Return Never returns.

Parameters

- arg_p: Pointer to the shell arguemnt struct struct `shell_args_t`. See the struct definition for a description of it's content.

struct **shell_history_elem_t**

#include <shell.h>

Public Members

```

struct shell_history_elem_t *next_p
struct shell_history_elem_t *prev_p
char buf[1]
struct shell_line_t

```

Public Members

```

char buf[CONFIG_SHELL_COMMAND_MAX]
int length
int cursor
struct shell_t

```

Public Members

```

void *chin_p
void *chout_p
void *arg_p
const char *name_p
const char *username_p
const char *password_p
struct shell_line_t line
struct shell_line_t prev_line
int carriage_return_received
int newline_received
int authorized
struct shell_history_elem_t *head_p
struct shell_history_elem_t *tail_p
struct shell_history_elem_t *current_p
struct shell_line_t pattern
struct shell_line_t match
int line_valid
struct circular_heap_t heap
uint8_t buf[CONFIG_SHELL_HISTORY_SIZE]
struct shell_t::@83::@84  shell_t::heap
struct shell_t::@83  shell_t::history

```

1.6.7 debug

The debug package on [Github](#).

harness — Test harness

In software testing, a test harness or automated test framework is a collection of software and test data configured to test a program unit by running it under varying conditions and monitoring its behavior and outputs. It has two main parts: the test execution engine and the test script repository.

This module implements the test execution engine.

The test scripts are part of the build system.

Example test suite

Below is an example of a test suite using the harness. It has three test cases; `test_passed`, `test_failed` and `test_skipped`.

The test macro `BTASSERT (condition)` should be used to validate conditions.

```
#include "simba.h"

static int test_passed(struct harness_t *harness_p)
{
    /* Return zero(0) when a test case passes. */
    return (0);
}

static int test_failed(struct harness_t *harness_p)
{
    /* Return a negative integer when a test case fails. BTASSERT
       will return -1 when the condition is false. */
    BTASSERT(0);

    return (0);
}

static int test_skipped(struct harness_t *harness_p)
{
    /* Return a positive integer when a test case is skipped. */
    return (1);
}

int main()
{
    /* Test harness and NULL terminated list of test cases.*/
    struct harness_t harness;
    struct harness_testcase_t harness_testcases[] = {
        { test_passed, "test_passed" },
        { test_failed, "test_failed" },
        { test_skipped, "test_skipped" },
        { NULL, NULL }
    };

    sys_start();
}
```

```

harness_init(&harness);
harness_run(&harness, harness_testcases);

return (0);
}

```

The output from the test suite is:

```

app:    test_suite-7.0.0 built 2016-07-25 17:38 CEST by erik.
board:  Linux
mcu:    Linux

enter: test_passed
exit: test_passed: PASSED

enter: test_failed
exit: test_failed: FAILED

enter: test_skipped
exit: test_skipped: SKIPPED

      NAME      STATE  PRIO   CPU  LOGMASK
      main      current    0    0%    0x0f
                   ready  127    0%    0x0f
harness report: total(3), passed(1), failed(1), skipped(1)

```

There are plenty of test suites in the [tst](#) folder on Github.

Source code: [src/debug/harness.h](#), [src/debug/harness.c](#)

Defines

BTASSERTN (cond, res, ...)

BTASSERT (cond, ...)

Assert given condition in a testcase. Print an error message and return -1 on error.

Typedefs

typedef int (***harness_testcase_cb_t**) (struct *harness_t* *harness_p)

The testcase function callback.

Return zero(0) if the testcase passed, a negative error code if the testcase failed, and a positive value if the testcase was skipped.

Parameters

- *harness_t*: The harness object.

Functions

int **harness_init** (struct *harness_t* *self_p)

Initialize given test harness.

Return zero(0) or negative error code.

Parameters

- self_p: Test harness to initialize.

int **harness_run** (struct *harness_t* *self_p, struct *harness_testcase_t* *testcases_p)

Run given testcases in given test harness.

Return zero(0) or negative error code.

Parameters

- self_p: Test harness.
- testcases_p: An array of testcases to run. The last element in the array must have `callback` and `name_p` set to NULL.

struct **harness_testcase_t**

Public Members

harness_testcase_cb_t **callback**

const char ***name_p**

struct **harness_t**

Public Members

struct uart_driver_t **uart**

log — Logging

The logging module consists of log objects and log handlers. A log object filters log entries and a log handler writes log entries to an output channel.

A log object called “log” and a log handler writing to standard output are created during the log module initialization. The log handler can be replaced by calling `log_set_default_handler_output_channel()`.

Normally one log object is created for each subsystem in an application. This gives the user the power to control which parts of the system to debug and/or monitor at runtime.

Sometimes it’s useful to write log entries to multiple channels. This is possible by creating and adding another log handler to the log module.

Log levels

There are five log levels defined; fatal, error, warning, info and debug. The log levels are defined as `LOG_<upper case level>` in the log module header file.

Log entry format

A log entry consists of a timestamp, log level, thread name, log object name and the message. The timestamp is the log entry creation time and the log level is one of fatal, error, warning, info and debug. The thread name is the name of the thread that created the log entry and the log object name is the name of the log object the entry was printed on. The message is a user defined string.

```
<timestamp>:<log level>:<thread name>:<log object name>: <message>
```

Debug file system commands

Three debug file system commands are available, all located in the directory `debug/log/`.

Command	Description
<code>list</code>	Print a list of all log objects.
<code>print <string></code>	Print a log entry using the default log object and log level <code>LOG_INFO</code> . This command has no use except to test that the log module works.
<code>set_log_mask <object> <mask></code>	Set the log mask to <code><mask></code> for log object <code><object></code> .

Example output from the shell:

```
$ debug/log/list
    OBJECT NAME  MASK
    default    0x0f
$ debug/log/print "Hello World!"
$ debug/log/set_log_mask default 0x1f
$ debug/log/list
    OBJECT NAME  MASK
    default    0x1f
$ debug/log/print "Hello World!!!"
56:info:main:default: Hello World!!!
```

Example

Here are a few example outputs using three log objects; *foo*, *bar* and the default log object *default*. All logs are from the main thread as can be seen in the third field in the entries.

```
23:info:main:foo: A foo info message.
24:info:main:bar: A bar info message.
37:debug:main:bar: A bar debug message.
56:error:main:default: A main error message.
```

Source code: `src/debug/log.h`, `src/debug/log.c`

Test code: `tst/debug/log/main.c`

Test coverage: `src/debug/log.c`

Defines

LOG_FATAL

LOG_ERROR

A handable error conditions.

LOG_WARNING

A warning.

LOG_INFO

Generic (useful) information about system operation.

LOG_DEBUG

Developer debugging messages.

LOG_MASK (level)

Create a log mask with given level set.

LOG_UPTO (level)

Set all levels up to and including given level.

LOG_ALL

Set all levels.

LOG_NONE

Clear all levels.

Functions

int **log_module_init** (void)

Initialize the logging module. This function must be called before calling any other function in this module.

The module will only be initialized once even if this function is called multiple times.

Return zero(0) or negative error code.

int **log_object_init** (struct *log_object_t* **self_p*, const char **name_p*, char *mask*)

Initialize given log object with given name and mask.

Return zero(0) or negative error code.

Parameters

- *self_p*: Log object to initialize.
- *name_p*: Log object name.
- *mask*: Log object mask.

int **log_object_set_log_mask** (struct *log_object_t* **self_p*, char *mask*)

Set given log mask for given log object.

Return zero(0) or negative error code.

Parameters

- *self_p*: Log object.
- *mask*: Log object mask.

char **log_object_get_log_mask** (struct *log_object_t* *self_p)

Get the log mask of given log object.

Return Log mask.

Parameters

- self_p: Log object.

int **log_object_is_enabled_for** (struct *log_object_t* *self_p, int level)

Check if given log level is enabled in given log object.

Return true(1) if given log level is enabled, false(0) if given log level is disabled, otherwise negative error code.

Parameters

- self_p: Log object, or NULL to check the level in the thread log mask.
- level: Log level to check.

int **log_object_print** (struct *log_object_t* *self_p, int level, const char *fmt_p, ...)

Check if given log level is set in the log object mask. If so, format a log entry and write it to all log handlers.

self_p may be NULL, and in that case the current thread's log mask is used instead of the log object mask.

Return zero(0) or negative error code.

Parameters

- self_p: Log object, or NULL to use the thread's log mask.
- level: Log level.
- fmt_p: Log format string.
- . . .: Variable argument list.

int **log_handler_init** (struct *log_handler_t* *self_p, void *chout_p)

Initialize given log handler with given output channel.

Return zero(0) or negative error code.

Parameters

- self_p: Log handler to initialize.
- chout_p: Output handler.

int **log_add_handler** (struct *log_handler_t* *handler_p)

Add given log handler to the list of log handlers. Log entries will be written to all log handlers in the list.

Return zero(0) or negative error code.

Parameters

- handler_p: Log handler to add.

int **log_remove_handler** (struct *log_handler_t* *handler_p)

Remove given log handler from the list of log handlers.

Return zero(0) or negative error code.

Parameters

- `handler_p`: Log handler to remove.

int **log_add_object** (struct *log_object_t* **object_p*)

Add given log object to the list of log objects. There are file system commands to list all log objects in the list and also modify their log mask.

Return zero(0) or negative error code.

Parameters

- `object_p`: Log object to add.

int **log_remove_object** (struct *log_object_t* **object_p*)

Remove given log object from the list of log objects.

Return zero(0) or negative error code.

Parameters

- `object_p`: Object to remove.

int **log_set_default_handler_output_channel** (void **chout_p*)

Set the output channel of the default log handler.

Return zero(0) or negative error code.

Parameters

- `chout_p`: Channel to set as the default output channel. May be NULL if no output should be written.

struct **log_handler_t**

Public Members

void ***chout_p**

struct *log_handler_t* ***next_p**

struct **log_object_t**

Public Members

const char ***name_p**

char **mask**

struct *log_object_t* ***next_p**

1.6.8 collections

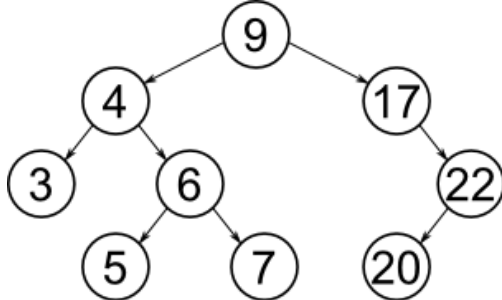
In computer science, a data structure is a particular way of organizing data in a computer so that it can be used efficiently.

The collections package on [Github](#).

binary_tree — Binary tree

A binary search tree consists of nodes, where each node has zero, one or two siblings. The left sibling has a lower value and the right sibling has a higher value than the parent.

Insert, delete and search operations all have the time complexity of $O(\log n)$.



Source code: [src/collections/binary_tree.h](#), [src/collections/binary_tree.c](#)

Test code: [tst/collections/binary_tree/main.c](#)

Test coverage: [src/collections/binary_tree.c](#)

Functions

int `binary_tree_init` (`struct binary_tree_t *self_p`)

Initialize given binary tree.

Return zero(0) or negative error code.

Parameters

- `self_p`: Binary tree.

int `binary_tree_insert` (`struct binary_tree_t *self_p`, `struct binary_tree_node_t *node_p`)

Insert given node into given binary tree.

There can not be two or more nodes in the tree with the same key. This function returns -1 if a node with the same key is already in the binary tree.

Return zero(0) on success, -1 if a node with the same key is already in the binary tree, otherwise negative error code.

Parameters

- `self_p`: Binary tree to insert the node into.
- `node_p`: Node to insert.

int `binary_tree_delete` (`struct binary_tree_t *self_p`, `int key`)

Delete given node from given binary tree.

Return zero(0) on success, -1 if the node was not found, otherwise negative error code.

Parameters

- `self_p`: Binary tree to delete the node from.
- `key`: Key of the node to delete.

struct `binary_tree_node_t` *`binary_tree_search` (**struct `binary_tree_t` *`self_p`**, `int key`)
Search the binary tree for the node with given key.

Return Pointer to found node or NULL if a node with given key was not found in the tree.

Parameters

- `self_p`: Binary tree to search in.
- `key`: Key of the binary tree node to search for.

void `binary_tree_print` (**struct `binary_tree_t` *`self_p`**)
Print given binary tree.

Parameters

- `self_p`: Binary tree to print.

struct `binary_tree_node_t`
`#include <binary_tree.h>`

Public Members

`int key`

`int height`

struct `binary_tree_node_t` *`left_p`

struct `binary_tree_node_t` *`right_p`

struct `binary_tree_t`

Public Members

struct `binary_tree_node_t` *`root_p`

bits — Bitwise operations

Source code: [src/collections/bits.h](#)

Test code: [tst/collections/bits/main.c](#)

Functions

static uint32_t `bits_insert_32` (`uint32_t dst`, `int position`, `int size`, `uint32_t src`)

fifo — First In First Out queuing

Source code: `src/collections/fifo.h`

Test code: `tst/collections/fifo/main.c`

Defines

FIFO_DEFINE_TEMPLATE (type)

Define the fifo structure and functions for a given type.

```

FIFO_DEFINE_TEMPLATE(int);

int foo()
{
    struct fifo_int_t fifo;
    int buf[4];
    int value;

    fifo_init_int(&fifo, buf, membersof(buf));

    // Put a value into the fifo.
    value = 10;
    fifo_put_int(&fifo, &value);

    // Get the value from the fifo.
    fifo_get_int(&fifo, &value);

    // Prints 'value = 10'.
    std_printf(FSTR("value= %d\r\n", value));
}

```

Parameters

- type: Type of the elements in the defined fifo.

Functions

static int fifo_init (struct *fifo_t* *self_p, int max)

Initialize given fifo.

Return zero(0) or negative error code.

Parameters

- self_p: Fifo to initialize.
- max: Maximum number of elements in the fifo.

static int fifo_put (struct *fifo_t* *self_p)

Put an element in the fifo.

Return Added element index in fifo, or -1 if there are no free positions.

Parameters

- `self_p`: Initialized fifo.

static int `fifo_get` (**struct** *fifo_t* *`self_p`)
Get the next element from the fifo.

Return The fetched element index in fifo , or -1 if the fifo is empty.

Parameters

- `self_p`: Initialized fifo.

struct `fifo_t`
#include <fifo.h>

Public Members

`int` `rdpos`
`int` `wrpos`
`void` *`buf_p`
`int` `max`

hash_map — Hash map

Source code: [src/collections/hash_map.h](#), [src/collections/hash_map.c](#)

Test code: [tst/collections/hash_map/main.c](#)

Test coverage: [src/collections/hash_map.c](#)

Typedefs

typedef `int` (*`hash_function_t`) (`long` key)

Functions

`int` `hash_map_init` (**struct** *hash_map_t* *`self_p`, **struct** *hash_map_bucket_t* *`buckets_p`, `size_t` `buckets_max`,
 struct *hash_map_entry_t* *`entries_p`, `size_t` `entries_max`, *hash_function_t* `hash`)
Initialize hash map with given parameters.

Return zero(0) or negative error code.

Parameters

- `self_p`: Initialized hash map.
- `buckets_p`: Array of buckets.
- `buckets_max`: Number of entries in `buckets_p`.
- `entries_p`: Array of empty entries.
- `entries_max`: Number of entries in `entries_p`.

- `hash`: Hash function.

int **hash_map_add** (struct *hash_map_t* **self_p*, long *key*, void **value_p*)

Add given key-value pair into hash map. Overwrites old value if the key is already present in map.

Return zero(0) or negative error code.

Parameters

- `self_p`: Initialized hash map.
- `key`: Key to hash.
- `value_p`: Value to insert for key.

int **hash_map_remove** (struct *hash_map_t* **self_p*, long *key*)

Remove given key from hash map.

Return zero(0) or negative error code.

Parameters

- `self_p`: Initialized hash map.
- `key`: Key to hash.

void ***hash_map_get** (struct *hash_map_t* **self_p*, long *key*)

Get value for given key.

Return Value for key or NULL if key was not found in the map.

Parameters

- `self_p`: Initialized hash map.
- `key`: Key to hash.

struct **hash_map_entry_t**

Public Members

struct *hash_map_entry_t* ***next_p**

long **key**

void ***value_p**

struct **hash_map_bucket_t**

Public Members

struct *hash_map_entry_t* ***list_p**

struct **hash_map_t**

Public Members

```
struct hash_map_bucket_t *buckets_p
size_t buckets_max
struct hash_map_entry_t *entries_p
hash_function_t hash
```

list — Abstract lists

Source code: <src/collections/list.h>

Defines

LIST_SL_INIT (list_p)
Initialize given singly linked list object.

Parameters

- list_p: List object to initialize.

LIST_SL_INIT_STRUCT

LIST_SL_PEEK_HEAD (list_p, element_pp)
Peek at the first element in the list.

Parameters

- list_p: List object.
- element_pp: First element of the list.

LIST_SL_ADD_HEAD (list_p, element_p)
Add given element to the beginning of given list.

Parameters

- list_p: List object.
- element_p: Element to add.

LIST_SL_ADD_TAIL (list_p, element_p)
Add given element to the end of given list.

Parameters

- list_p: List object.
- element_p: Element to add.

LIST_SL_REMOVE_HEAD (list_p, element_pp)
Get the first element of given list and then remove it from given list.

Parameters

- `list_p`: List object.
- `element_pp`: First element of the list.

LIST_SL_ITERATOR_INIT (`iterator_p`, `list_p`)
Initialize given iterator object.

Parameters

- `iterator_p`: Iterator to initialize.
- `list_p`: List object to iterate over.

LIST_SL_ITERATOR_NEXT (`iterator_p`, `element_pp`)
Get the next element from given iterator object.

Parameters

- `iterator_p`: Iterator object.
- `element_pp`: Next element of the list.

LIST_SL_REMOVE_ELEM (`list_p`, `iterator_p`, `element_p`, `iterator_element_p`, `previous_element_p`)
Remove given element from given list.

Parameters

- `list_p`: List object.
- `iterator_p`: Used internally.
- `element_p`: Used internally.
- `iterator_element_p`: Used internally.
- `previous_element_p`: Used internally.

struct list_next_t
#include <list.h>

Public Members

`void *next_p`

struct list_singly_linked_t

Public Members

`void *head_p`

`void *tail_p`

struct list_sl_iterator_t

Public Members

`void *next_p`

1.6.9 alloc

Memory management is the act of managing computer memory. The essential requirement of memory management is to provide ways to dynamically allocate portions of memory to programs at their request, and free it for reuse when no longer needed.

The alloc package on [Github](#).

circular_heap — Circular heap

The circular heap is a dynamic memory allocator allocating buffers in a circular buffer. This puts a restriction on the user to free allocated buffers in the same order as they were allocated. This allocator is useful if you know the allocation order and need a low memory overhead on each allocated buffer and no memory fragmentation.

Below is an example of the internal state of a circular heap when buffers are allocated and freed.

1. After initialization *begin*, *alloc* and *free* have the same value. All memory is available for allocation.

```
begin
alloc
free                                     end
|-----|
```

2. Allocating a buffer increments *alloc*.

```
begin
free                                     alloc                                     end
|=====|-----|
```

3. Allocating another buffer increments *alloc* once again.

```
begin
free                                     alloc                                     end
|=====|-----|
```

4. Freeing the first buffer increments *free* to the position of the first *alloc*.

```
begin                                     free                                     alloc                                     end
|-----|=====|-----|
```

5. Allocating a buffer that is bigger than the available space between *alloc* and *end* results in a buffer starting at *begin*. The memory between the old *alloc* and *end* will be unused.

```
begin                                     alloc   free                                     end
|=====|-----|=====|ooooooooo|
```

6. Freeing the second buffer increments *free* to the position of the second *alloc*.

```
begin                                     alloc                                     free                                     end
|=====|-----|ooooooooo|
```

7. Freeing the third buffer sets *free* to *alloc*. All memory is available for allocation once again.

```
begin                                     alloc
free                                     end
|-----|-----|
```

8. Done!

Source code: `src/alloc/circular_heap.h`, `src/alloc/circular_heap.c`

Test code: `tst/alloc/circular_heap/main.c`

Test coverage: `src/alloc/circular_heap.c`

Functions

int **circular_heap_init** (struct *circular_heap_t* *self_p, void *buf_p, size_t size)

Initialize given circular_heap.

Return zero(0) or negative error code.

Parameters

- self_p: Circular heap to initialize.
- buf_p: Memory buffer.
- size: Size of the memory buffer.

void ***circular_heap_alloc** (struct *circular_heap_t* *self_p, size_t size)

Allocate a buffer of given size from given circular heap.

Return Pointer to allocated buffer, or NULL on failure.

Parameters

- self_p: Circular heap to allocate from.
- size: Number of bytes to allocate.

int **circular_heap_free** (struct *circular_heap_t* *self_p, void *buf_p)

Free the oldest allocated buffer.

Return zero(0) or negative error code.

Parameters

- self_p: Circular heap to free to.
- buf_p: Buffer to free. Must be the oldest allocated buffer.

struct **circular_heap_t**
#include <circular_heap.h>

Public Members

void ***begin_p**

void ***end_p**

void ***alloc_p**

void ***free_p**

heap — Heap

Source code: `src/alloc/heap.h`, `src/alloc/heap.c`

Test code: `tst/alloc/heap/main.c`

Test coverage: `src/alloc/heap.c`

Defines

HEAP_FIXED_SIZES_MAX

Functions

int **heap_init** (struct *heap_t* **self_p*, void **buf_p*, size_t *size*, size_t *sizes*[HEAP_FIXED_SIZES_MAX])
Initialize given heap.

Return zero(0) or negative error code.

Parameters

- *self_p*: Heap to initialize.
- *buf_p*: Heap memory buffer.
- *size*: Size of the heap memory buffer.

void ***heap_alloc** (struct *heap_t* **self_p*, size_t *size*)
Allocate a buffer of given size from given heap.

Return Pointer to allocated buffer, or NULL on failure.

Parameters

- *self_p*: Heap to allocate from.
- *size*: Number of bytes to allocate.

int **heap_free** (struct *heap_t* **self_p*, void **buf_p*)
Decrement the share count by once and free the buffer if the count becomes zero(0).

Return Share count after the free, or negative error code.

Parameters

- *self_p*: Heap of given buffer.
- *buf_p*: Memory buffer to free.

int **heap_share** (struct *heap_t* **self_p*, const void **buf_p*, int *count*)
Share given buffer *count* times.

Return zero(0) or negative error code.

Parameters

- *self_p*: Heap of given buffer.

- `buf_p`: Buffer to share.
- `count`: Share count.

`struct heap_fixed_t`

Public Members

`void *free_p`

`size_t size`

`struct heap_dynamic_t`

Public Members

`void *free_p`

`struct heap_t`

Public Members

`void *buf_p`

`size_t size`

`void *next_p`

`struct heap_fixed_t fixed[HEAP_FIXED_SIZES_MAX]`

`struct heap_dynamic_t dynamic`

1.6.10 text

Text parsing, editing and colorization.

The text package on [Github](#).

color — ANSI colors

Source code: [src/text/color.h](#)

Defines

`COLOR_RESET`

`COLOR_BOLD_ON`

`COLOR_ITALICS_ON`

`COLOR_UNDERLINE_ON`

`COLOR_INVERSE_ON`

COLOR_STRIKETHROUGH_ON
COLOR_BOLD_OFF
COLOR_ITALICS_OFF
COLOR_UNDERLINE_OFF
COLOR_INVERSE_OFF
COLOR_STRIKETHROUGH_OFF
COLOR_FOREGROUND_BLACK
COLOR_FOREGROUND_RED
COLOR_FOREGROUND_GREEN
COLOR_FOREGROUND_YELLOW
COLOR_FOREGROUND_BLUE
COLOR_FOREGROUND_MAGENTA
COLOR_FOREGROUND_CYAN
COLOR_FOREGROUND_WHITE
COLOR_FOREGROUND_DEFAULT
COLOR_BACKGROUND_BLACK
COLOR_BACKGROUND_RED
COLOR_BACKGROUND_GREEN
COLOR_BACKGROUND_YELLOW
COLOR_BACKGROUND_BLUE
COLOR_BACKGROUND_MAGENTA
COLOR_BACKGROUND_CYAN
COLOR_BACKGROUND_WHITE
COLOR_BACKGROUND_DEFAULT
COLOR (...)

configfile — Configuration file (INI-file)

The INI file format is an informal standard for configuration files for some platforms or software. INI files are simple text files with a basic structure composed of sections, properties, and values.

More information on [Wikipedia](#).

File format description

- Line terminators: `\n`, `\r\n` or `\n\r`.
- Opening bracket (`[`) at the beginning of a line indicates a section. The section name is all characters until a closing bracket (`]`).
- A property line starts with its name, then a colon (`:`) or equal sign (`=`), and then the value.

- Semicolon (;) or number sign (#) at the beginning of a line indicate a comment.

Example file

```
; last modified 1 April 2001 by John Doe
[owner]
name = John Doe
organization = Acme Widgets Inc.

[database]
; use IP address in case network name resolution is not working
server = 192.0.2.62
port = 143
file = "payroll.dat"
```

Source code: `src/text/configfile.h`, `src/text/configfile.c`

Test code: `tst/text/configfile/main.c`

Test coverage: `src/text/configfile.c`

Functions

int **configfile_init** (struct *configfile_t* *self_p, char *buf_p, size_t size)

Initialize given configuration file object.

Return zero(0) or negative error code.

Parameters

- self_p: Object to initialize.
- buf_p: Configuration file contents as a NULL terminated string.
- size: Size of the configuration file contents.

int **configfile_set** (struct *configfile_t* *self_p, const char *section_p, const char *property_p, const char *value_p)

Set the value of given property in given section.

Return zero(0) or negative error code.

Parameters

- self_p: Initialized parser.
- section_p: Section to set the property from.
- property_p: Property to set the value for.
- value_p: NULL terminated value to set.

char ***configfile_get** (struct *configfile_t* *self_p, const char *section_p, const char *property_p, char *value_p, int length)

Get the value of given property in given section.

Return Value pointer or NULL on failure.

Parameters

- `self_p`: Initialized parser.
- `section_p`: Section to get the property from.
- `property_p`: Property to get the value for.
- `value_p`: Value of given property in given section.
- `size`: Size of the value buffer.

int **configfile_get_long** (struct *configfile_t* **self_p*, const char **section_p*, const char **property_p*,
long **value_p*)

Get the value of given property in given section, converted to an integer.

Return zero(0) or negative error code.

Parameters

- `self_p`: Initialized parser.
- `section_p`: Section to get the property from.
- `property_p`: Property to get the value for.
- `value_p`: Value of given property in given section.

int **configfile_get_float** (struct *configfile_t* **self_p*, const char **section_p*, const char **property_p*,
float **value_p*)

Get the value of given property in given section, converted to a float.

Return zero(0) or negative error code.

Parameters

- `self_p`: Initialized parser.
- `section_p`: Section to get the property from.
- `property_p`: Property to get the value for.
- `value_p`: Value of given property in given section.

struct **configfile_t**
#include <configfile.h>

Public Members

char ***buf_p**

size_t **size**

re — Regular expressions

Source code: [src/text/re.h](#), [src/text/re.c](#)

Test code: [tst/text/re/main.c](#)

Test coverage: [src/text/re.c](#)

Defines

RE_IGNORECASE

RE_DOTALL

Make the '.' special character match any character at all, including a newline; without this flag, '.' will match anything except a newline.

RE_MULTILINE

When specified, the pattern character '^' matches at the beginning of the string and at the beginning of each line (immediately following each newline); and the pattern character '\$' matches at the end of the string and at the end of each line (immediately preceding each newline). By default, '^' matches only at the beginning of the string, and '\$' only at the end of the string and immediately before the newline (if any) at the end of the string.

Functions

char ***re_compile** (char **compiled_p*, const char **pattern_p*, char *flags*, size_t *size*)

Compile given pattern.

Pattern syntax:

- '.' - Any character.
- '^' - Beginning of the string (**not yet supported**).
- '\$' - End of the string (**not yet supported**).
- '?' - Zero or one repetitions (greedy).
- '*' - Zero or more repetitions (greedy).
- '+' - One or more repetitions (greedy).
- '???' - Zero or one repetitions (non-greedy).
- '*?' - Zero or more repetitions (non-greedy).
- '+?' - One or more repetitions (non-greedy).
- '{m}' - Exactly m repetitions.
- '\\' - Escape character.
- '[' - Set of characters.
- '|' - Alternatives (**not yet supported**).
- '(...)' - Groups (**not yet supported**).
- '\\d' - Decimal digits [0-9].
- '\\w' - Alphanumeric characters [a-zA-Z0-9_].
- '\\s' - Whitespace characters [\t\r\n\f\v].

Return Compiled patten, or NULL if the compilation failed.

Parameters

- *compiled_p*: Compiled regular expression pattern.

- `pattern_p`: Regular expression pattern.
- `flags`: A combination of the flags `RE_IGNORECASE`, `RE_DOTALL` and `RE_MULTILINE` (`RE_MULTILINE` is **not yet supported**).
- `size`: Size of the compiled buffer.

`ssize_t re_match(const char *compiled_p, const char *buf_p, size_t size, struct re_group_t *groups_p, size_t *number_of_groups_p)`

Apply given regular expression to the beginning of given string.

Return Number of matched bytes or negative error code.

Parameters

- `compiled_p`: Compiled regular expression pattern. Compile a pattern with `re_compile()`.
- `buf_p`: Buffer to apply the compiled pattern to.
- `size`: Number of bytes in the buffer.
- `groups_p`: Read groups or NULL.
- `number_of_groups_p`: Number of read groups or NULL.

`struct re_group_t`

Public Members

`const char *buf_p`

`ssize_t size`

std — Standard functions

Source code: [src/text/std.h](#), [src/text/std.c](#)

Test code: [tst/text/std/main.c](#)

Test coverage: [src/text/std.c](#)

Functions

`int std_module_init(void)`

`ssize_t std_sprintf(char *dst_p, FAR const char *fmt_p, ...)`

Format and write data to destination buffer. The buffer must be big enough to fit the formatted string. The output is null terminated.

A format specifier has this format:

`%[flags][width][length]specifier`

where

- `flags`: 0 or `-`
- `width`: 0..127

- length: l for long or nothing
- specifier: c, s, d, i, u, x or f

Return Length of the string written to the destination buffer, not including the null termination, or negative error code.

Parameters

- dst_p: Destination buffer. The formatted string is written to this buffer.
- fmt_p: Format string.
- ...: Variable arguments list.

ssize_t std_snprintf(char * dst_p, size_t size, FAR const char * fmt_p, ...)

Format and write data to given buffer. The output is null terminated.

Return Length of the string written to the destination buffer, not including the null termination, or negative error code.

Parameters

- dst_p: Destination buffer. The formatted string is written to this buffer.
- size: Size of the destination buffer.
- fmt_p: Format string.
- ...: Variable arguments list.

ssize_t std_vsprintf(char * dst_p, FAR const char * fmt_p, va_list * ap_p)

Format and write data to given buffer. The output is null terminated.

Return Length of the string written to the destination buffer, not including the null termination, or negative error code.

Parameters

- dst_p: Destination buffer. The formatted string is written to this buffer.
- fmt_p: Format string.
- ap_p: Variable arguments list.

ssize_t std_vsnprintf(char * dst_p, size_t size, FAR const char * fmt_p, va_list * ap_p)

Format and write data to given buffer. The output is null terminated.

Return Length of the string written to the destination buffer, not including the null termination, or negative error code.

Parameters

- dst_p: Destination buffer. The formatted string is written to this buffer.
- size: Size of the destination buffer.
- fmt_p: Format string.
- ap_p: Variable arguments list.

ssize_t std_printf(far_string_t fmt_p, ...)

Format and print data to standard output. The output is not null terminated.

See `std_sprintf()` for the the format string specification.

Return Number of characters written to standard output, or negative error code.

Parameters

- `fmt_p`: Format string.
- `...`: Variable arguments list.

ssize_t std_vprintf(FAR const char * `fmt_p`, va_list * `ap_p`)

Format and print data to standard output. The output is not null terminated.

See `std_sprintf()` for the the format string specification.

Return Number of characters written to standard output, or negative error code.

Parameters

- `fmt_p`: Format string.
- `ap_p`: Variable arguments list.

ssize_t std_fprintf(void * `chan_p`, FAR const char * `fmt_p`, ...)

Format and print data to channel. The output is not null terminated.

See `std_sprintf()` for the the format string specification.

Return Number of characters written to given channel, or negative error code.

Parameters

- `chan_p`: Output channel.
- `fmt_p`: Format string.
- `...`: Variable arguments list.

ssize_t std_vfprintf(void * `chan_p`, FAR const char * `fmt_p`, va_list * `ap_p`)

Format and print data to channel. The output is not null terminated.

See `std_sprintf()` for the the format string specification.

Return Number of characters written to given channel, or negative error code.

Parameters

- `chan_p`: Output channel.
- `fmt_p`: Format string.
- `...`: Variable arguments list.

const char *std_strtol (const char *`str_p`, long *`value_p`)

Convert string to integer.

Return Pointer to the next byte or NULL on failure.

Parameters

- `str_p`: Integer string.
- `value_p`: Integer value.

int std_strcpy(char * `dst_p`, FAR const char * `src_p`)

Copy string from far memory to memory.

Return String length or negative error code.

Parameters

- `dst_p`: Normal memory string.
- `src_p`: Far memory string.

int std_strcmp(const char * str_p, FAR const char * fstr_p)

Compare a string with a far string.

Return zero(0) if match, otherwise the difference of the mismatched characters

Parameters

- `str_p`: Normal memory string.
- `fstr_p`: Far memory string.

int std_strcmp_f(FAR const char * fstr0_p, FAR const char * fstr1_p)

Compare two far strings.

Return zero(0) if match, otherwise the difference of the mismatched characters.

Parameters

- `fstr0_p`: Far memory string.
- `fstr1_p`: Far memory string.

int std_strncmp(FAR const char * fstr_p, const char * str_p, size_t size)

Compare at most `size` bytes of one far string and one string.

Return zero(0) if match, otherwise the difference of the mismatched characters.

Parameters

- `fstr_p`: Far memory string.
- `str_p`: String.
- `size`: Compare at most `size` number of bytes.

int std_strncmp_f(FAR const char * fstr0_p, FAR const char * fstr1_p, size_t size)

Compare at most `size` bytes of two far strings.

Return zero(0) if match, otherwise the difference of the mismatched characters.

Parameters

- `fstr0_p`: Far memory string.
- `fstr1_p`: Far memory string.
- `size`: Compare at most `size` number of bytes.

int std_strlen(FAR const char * fstr_p)

Get the length in bytes of given far string, not including null termination.

Return String length in number of bytes (not including the null termination).

Parameters

- `fstr_p`: Far memory string.

char *std_strip(char *str_p, const char *strip_p)

Strip leading and trailing characters from a string. The characters to strip are given by `strip_p`.

Return Pointer to the stripped string.

Parameters

- `str_p`: String to strip characters from.
- `strip_p`: Characters to strip or NULL for whitespace characters. Must be null-terminated.

1.6.11 encode

In computing, a character encoding is used to represent a repertoire of characters by some kind of an encoding system. The encode package on [Github](#).

base64 — Base64 encoding and decoding.

Source code: [src/encode/base64.h](#), [src/encode/base64.c](#)

Test code: [tst/encode/base64/main.c](#)

Test coverage: [src/encode/base64.c](#)

Functions

int **base64_encode** (char **dst_p*, const void **src_p*, size_t *size*)

int **base64_decode** (void **dst_p*, const char **src_p*, size_t *size*)

Decode given base64 encoded buffer. The decoded data will be ~25% smaller than the destination data. Choose the destination buffer size accordingly.

Return zero(0) or negative error code.

Parameters

- `dst_p`: Output data.
- `src_p`: Encoded input data.
- `size`: Number of bytes in the encoded input data.

json — JSON encoding and decoding

Source code: [src/encode/json.h](#), [src/encode/json.c](#)

Test code: [tst/encode/json/main.c](#)

Test coverage: [src/encode/json.c](#)

Enums

enum `json_type_t`

Values:

`JSON_UNDEFINED = 0`

Undefined type.

`JSON_OBJECT = 1`

Object, `{ }`.

`JSON_ARRAY = 2`

Array, `[]`.

`JSON_STRING = 3`

String, `"..."`.

`JSON_PRIMITIVE = 4`

Other primitive: number, boolean (true/false) or null.

enum `json_err_t`

Values:

`JSON_ERROR_NOMEM = -1`

Not enough tokens were provided.

`JSON_ERROR_INVAL = -2`

Invalid character inside JSON string.

`JSON_ERROR_PART = -3`

The string is not a full JSON packet, more bytes expected.

Functions

int `json_init (struct json_t *self_p, struct json_tok_t *tokens_p, int num_tokens)`

Initialize given JSON object. The JSON object must be initialized before it can be used to parse and dump JSON data.

Return zero(0) or negative error code.

Parameters

- `self_p`: JSON object to initialize.
- `tokens_p`: Array of tokens. The tokens are either filled by the parsing function `json_parse()`, or already filled by the user when calling this function. The latter can be used to dump the tokens as a string by calling `json_dump()` or `json_dumps()`.
- `num_tokens`: Number of tokens in the array.

int `json_parse (struct json_t *self_p, const char *js_p, size_t len)`

Parse given JSON data string into an array of tokens, each describing a single JSON object.

Return Number of decoded tokens or negative error code.

Parameters

- `self_p`: JSON object.
- `js_p`: JSON string to parse.

- `len`: JSON string length in bytes.

`ssize_t json_dumps (struct json_t *self_p, struct json_tok_t *tokens_p, char *js_p)`
Format and write given JSON tokens into a string.

Return Dumped string length (not including termination) or negative error code.

Parameters

- `self_p`: JSON object.
- `tokens_p`: Root token to dump. Set to NULL to dump the whole object.
- `js_p`: Dumped null terminated JSON string.

`ssize_t json_dump (struct json_t *self_p, struct json_tok_t *tokens_p, void *out_p)`
Format and write given JSON tokens to given channel.

Return Dumped string length (not including termination) or negative error code.

Parameters

- `self_p`: JSON object.
- `tokens_p`: Root token to dump. Set to NULL to dump the whole object.
- `out_p`: Channel to dump the null terminated JSON string to.

`struct json_tok_t *json_root (struct json_t *self_p)`
Get the root token.

Return The root token or NULL on failure.

Parameters

- `self_p`: JSON object.

`struct json_tok_t *json_object_get (struct json_t *self_p, const char *key_p, struct json_tok_t *object_p)`
Get the value the string token with given key.

Return Token or NULL on error.

Parameters

- `self_p`: JSON object.
- `key_p`: Key of the value to get.
- `object_p`: The object to get the value from.

`struct json_tok_t *json_object_get_primitive (struct json_t *self_p, const char *key_p, struct json_tok_t *object_p)`
Get the value of the primitive token with given key.

Return Token or NULL on error.

Parameters

- `self_p`: JSON object.
- `key_p`: Key of the value to get.

- `object_p`: The object to get the value from.

struct `json_tok_t` *`json_array_get` (struct `json_t` *`self_p`, int `index`, struct `json_tok_t` *`array_p`)
Get the token of given array index.

Return Token or NULL on error.

Parameters

- `self_p`: JSON object.
- `index`: Index to get.
- `array_p`: The array to get the element from.

void `json_token_object` (struct `json_tok_t` *`token_p`, int `num_keys`)
Initialize a JSON object token.

Parameters

- `token_p`: Initialized token.
- `num_keys`: Number of keys in the object.

void `json_token_array` (struct `json_tok_t` *`token_p`, int `num_elements`)
Initialize a JSON array token.

Parameters

- `token_p`: Initialized token.
- `num_elements`: Number of array elements.

void `json_token_true` (struct `json_tok_t` *`token_p`)
Initialize a JSON boolean true token.

Parameters

- `token_p`: Initialized token.

void `json_token_false` (struct `json_tok_t` *`token_p`)
Initialize a JSON boolean false token.

Parameters

- `token_p`: Initialized token.

void `json_token_null` (struct `json_tok_t` *`token_p`)
Initialize a JSON null token.

Parameters

- `token_p`: Initialized token.

void `json_token_number` (struct `json_tok_t` *`token_p`, const char *`buf_p`, size_t `size`)
Initialize a JSON number (integer/float) token.

Parameters

- `token_p`: Initialized token.

- `buf_p`: Number as a string.
- `size`: String length.

void **json_token_string** (struct *json_tok_t* *`token_p`, const char *`buf_p`, size_t `size`)
Initialize a JSON string token.

Parameters

- `token_p`: Initialized token.
- `buf_p`: String.
- `size`: String length.

struct **json_tok_t**

Public Members

json_type_t **type**

const char ***buf_p**

size_t **size**

int **num_tokens**

struct **json_t**

Public Members

unsigned int **pos**

Offset in the JSON string.

unsigned int **toknext**

Next token to allocate.

int **toksuper**

Superior token node, e.g parent object or array.

struct *json_tok_t* ***tokens_p**

Array of tokens.

int **num_tokens**

Number of tokens in the tokens array.

1.6.12 hash

A hash function is any function that can be used to map data of arbitrary size to data of fixed size.

The hash package on [Github](#).

crc — Cyclic Redundancy Checks

Source code: [src/hash/crc.h](#), [src/hash/crc.c](#)

Test code: [tst/hash/crc/main.c](#)

Test coverage: [src/hash/crc.c](#)

Functions

`uint32_t crc_32 (uint32_t crc, const void *buf_p, size_t size)`

`uint16_t crc_ccitt (uint16_t crc, const void *buf_p, size_t size)`

Calculate a 16 bits crc using the CCITT algorithm (polynomial $x^{16}+x^{12}+x^5+x^1$).

Return Calculated crc.

Parameters

- `crc`: Initial crc. Should be 0xffff for CCITT.
- `buf_p`: Buffer to calculate crc of.
- `size`: Size of the buffer.

`uint16_t crc_xmodem (uint16_t crc, const void *buf_p, size_t size)`

Calculate a 16 bits crc using the XModem algorithm (polynomial $x^{16}+x^{12}+x^5+x^1$).

Return Calculated crc.

Parameters

- `crc`: Initial crc. Should be 0x0000 for XModem.
- `buf_p`: Buffer to calculate crc of.
- `size`: Size of the buffer.

`uint8_t crc_7 (const void *buf_p, size_t size)`

Calculate a 8 bits crc using the CRC-7 algorithm (polynomial x^7+x^3+1).

Return Calculated crc.

Parameters

- `buf_p`: Buffer to calculate crc of.
- `size`: Size of the buffer.

sha1 — SHA1

Source code: [src/hash/sha1.h](#), [src/hash/sha1.c](#)

Test code: [tst/hash/main.c](#)

Test coverage: [src/hash/sha1.c](#)

Functions

int **sha1_init** (struct *sha1_t* **self_p*)

Initialize given SHA1 object.

Return zero(0) or negative error code.

Parameters

- *self_p*: SHA1 object.

int **sha1_update** (struct *sha1_t* **self_p*, void **buf_p*, size_t *size*)

Update the sha object with the given buffer. Repeated calls are equivalent to a single call with the concatenation of all the arguments.

Return zero(0) or negative error code.

Parameters

- *self_p*: SHA1 object.
- *buf_p*: Buffer to update the sha object with.
- *size*: Size of the buffer.

int **sha1_digest** (struct *sha1_t* **self_p*, uint8_t **hash_p*)

Return the digest of the strings passed to the sha1_update() method so far. This is a 20-byte value which may contain non-ASCII characters, including null bytes.

Return zero(0) or negative error code.

Parameters

- *self_p*: SHA1 object.
- *hash_p*: Hash sum.

struct **sha1_t**

#include <sha1.h>

Public Members

uint8_t **buf**[64]

uint32_t **size**

struct **sha1_t::@33** **sha1_t::block**

uint32_t **h**[5]

uint64_t **size**

1.6.13 multimedia

The multimedia package on [Github](#).

midi — Musical Instrument Digital Interface

Source code: `src/multimedia/midi.h`, `src/multimedia/midi.c`

Test code: `tst/multimedia/midi/main.c`

Test coverage: `src/multimedia/midi.c`

Defines

`MIDI_BAUDRATE`
`MIDI_NOTE_OFF`
`MIDI_NOTE_ON`
`MIDI_POLYPHONIC_KEY_PRESSURE`
`MIDI_CONTROL_CHANGE`
`MIDI_PROGRAM_CHANGE`
`MIDI_CHANNEL_PRESSURE`
`MIDI_PITCH_BEND_CHANGE`
`MIDI_SET_INTRUMENT`
`MIDI_PERC`
`MIDI_NOTE_MAX`
`MIDI_NOTE_A0`
`MIDI_NOTE_B0`
`MIDI_NOTE_C1`
`MIDI_NOTE_D1`
`MIDI_NOTE_E1`
`MIDI_NOTE_F1`
`MIDI_NOTE_G1`
`MIDI_NOTE_A1`
`MIDI_NOTE_B1`
`MIDI_NOTE_C2`
`MIDI_NOTE_D2`
`MIDI_NOTE_E2`
`MIDI_NOTE_F2`
`MIDI_NOTE_G2`
`MIDI_NOTE_A2`
`MIDI_NOTE_B2`
`MIDI_NOTE_C3`

MIDI_NOTE_D3
MIDI_NOTE_E3
MIDI_NOTE_F3
MIDI_NOTE_G3
MIDI_NOTE_A3
MIDI_NOTE_B3
MIDI_NOTE_C4
MIDI_NOTE_D4
MIDI_NOTE_E4
MIDI_NOTE_F4
MIDI_NOTE_G4
MIDI_NOTE_A4
MIDI_NOTE_B4
MIDI_NOTE_C5
MIDI_NOTE_D5
MIDI_NOTE_E5
MIDI_NOTE_F5
MIDI_NOTE_G5
MIDI_NOTE_A5
MIDI_NOTE_B5
MIDI_NOTE_C6
MIDI_NOTE_D6
MIDI_NOTE_E6
MIDI_NOTE_F6
MIDI_NOTE_G6
MIDI_NOTE_A6
MIDI_NOTE_B6
MIDI_NOTE_C7
MIDI_NOTE_D7
MIDI_NOTE_E7
MIDI_NOTE_F7
MIDI_NOTE_G7
MIDI_NOTE_A7
MIDI_NOTE_B7
MIDI_NOTE_C8
MIDI_PERC_ACOUSTIC_BASS_DRUM

MIDI_PERC_BASS_DRUM_1
MIDI_PERC_SIDE_STICK
MIDI_PERC_ACOUSTIC_SNARE
MIDI_PERC_HAND_CLAP
MIDI_PERC_ELECTRIC_SNARE
MIDI_PERC_LOW_FLOOR_TOM
MIDI_PERC_CLOSED_HI_HAT
MIDI_PERC_HIGH_FLOOR_TOM
MIDI_PERC_PEDAL_HI_HAT
MIDI_PERC_LOW_TOM
MIDI_PERC_OPEN_HI_HAT
MIDI_PERC_LOW_MID_TOM
MIDI_PERC_HI_MID_TOM
MIDI_PERC_CRASH_CYMBAL_1
MIDI_PERC_HIGH_TOM
MIDI_PERC_RIDE_CYMBAL_1
MIDI_PERC_CHINESE_CYMBAL
MIDI_PERC_RIDE_BELL
MIDI_PERC_TAMBOURINE
MIDI_PERC_SPLASH_CYMBAL
MIDI_PERC_COWBELL
MIDI_PERC_CRASH_CYMBAL_2
MIDI_PERC_VIBRASLAP
MIDI_PERC_RIDE_CYMBAL_2
MIDI_PERC_HI_BONGO
MIDI_PERC_LOW_BONGO
MIDI_PERC_MUTE_HI_CONGA
MIDI_PERC_OPEN_HI_CONGA
MIDI_PERC_LOW_CONGA
MIDI_PERC_HIGH_TIMBALE
MIDI_PERC_LOW_TIMBALE
MIDI_PERC_HIGH_AGOGO
MIDI_PERC_LOW_AGOGO
MIDI_PERC_CABASA
MIDI_PERC_MARACAS
MIDI_PERC_SHORT_WHISTLE

`MIDI_PERC_LONG_WHISTLE`
`MIDI_PERC_SHORT_GUIRO`
`MIDI_PERC_LONG_GUIRO`
`MIDI_PERC_CLAVES`
`MIDI_PERC_HI_WOOD_BLOCK`
`MIDI_PERC_LOW_WOOD_BLOCK`
`MIDI_PERC_MUTE_CUICA`
`MIDI_PERC_OPEN_CUICA`
`MIDI_PERC_MUTE_TRIANGLE`
`MIDI_PERC_OPEN_TRIANGLE`

Functions

float `midi_note_to_frequency` (int *note*)
Get the frequency for given note.

Return Note frequency.

Parameters

- `note`: MIDI note.

1.6.14 boards

The boards supported by *Simba*.

The boards on [Github](#).

`arduino_due` — Arduino Due

Source code: [src/boards/arduino_due/board.h](#), [src/boards/arduino_due/board.c](#)

Hardware reference: [Arduino Due](#)

Defines

`pin_d0_dev`
`pin_d1_dev`
`pin_d2_dev`
`pin_d3_dev`
`pin_d4_dev`
`pin_d5_dev`
`pin_d6_dev`

pin_d7_dev
pin_d8_dev
pin_d9_dev
pin_d10_dev
pin_d11_dev
pin_d12_dev
pin_d13_dev
pin_d14_dev
pin_d15_dev
pin_d16_dev
pin_d17_dev
pin_d18_dev
pin_d19_dev
pin_d20_dev
pin_d21_dev
pin_d22_dev
pin_d23_dev
pin_d24_dev
pin_d25_dev
pin_d26_dev
pin_d27_dev
pin_d28_dev
pin_d29_dev
pin_d30_dev
pin_d31_dev
pin_d32_dev
pin_d33_dev
pin_d34_dev
pin_d35_dev
pin_d36_dev
pin_d37_dev
pin_d38_dev
pin_d39_dev
pin_d40_dev
pin_d41_dev
pin_d42_dev

pin_d43_dev
pin_d44_dev
pin_d45_dev
pin_d46_dev
pin_d47_dev
pin_d48_dev
pin_d49_dev
pin_d50_dev
pin_d51_dev
pin_d52_dev
pin_d53_dev
pin_a0_dev
pin_a1_dev
pin_a2_dev
pin_a3_dev
pin_a4_dev
pin_a5_dev
pin_a6_dev
pin_a7_dev
pin_a8_dev
pin_a9_dev
pin_a10_dev
pin_a11_dev
pin_led_dev
pin_dac0_dev
pin_dac1_dev
exti_d0_dev
exti_d1_dev
exti_d2_dev
exti_d3_dev
exti_d4_dev
exti_d5_dev
exti_d6_dev
exti_d7_dev
exti_d8_dev
exti_d9_dev

exti_d10_dev
exti_d11_dev
exti_d12_dev
exti_d13_dev
exti_d14_dev
exti_d15_dev
exti_d16_dev
exti_d17_dev
exti_d18_dev
exti_d19_dev
exti_d20_dev
exti_d21_dev
exti_d22_dev
exti_d23_dev
exti_d24_dev
exti_d25_dev
exti_d26_dev
exti_d27_dev
exti_d28_dev
exti_d29_dev
exti_d30_dev
exti_d31_dev
exti_d32_dev
exti_d33_dev
exti_d34_dev
exti_d35_dev
exti_d36_dev
exti_d37_dev
exti_d38_dev
exti_d39_dev
exti_d40_dev
exti_d41_dev
exti_d42_dev
exti_d43_dev
exti_d44_dev
exti_d45_dev

exti_d46_dev
exti_d47_dev
exti_d48_dev
exti_d49_dev
exti_d50_dev
exti_d51_dev
exti_d52_dev
exti_d53_dev
exti_a0_dev
exti_a1_dev
exti_a2_dev
exti_a3_dev
exti_a4_dev
exti_a5_dev
exti_a6_dev
exti_a7_dev
exti_a8_dev
exti_a9_dev
exti_a10_dev
exti_a11_dev
exti_led_dev
exti_dac0_dev
exti_dac1_dev
pwm_d2_dev
pwm_d3_dev
pwm_d5_dev
pwm_d6_dev
pwm_d7_dev
pwm_d8_dev
pwm_d9_dev
pwm_d10_dev
pwm_d11_dev
pwm_d12_dev
adc_0_dev
dac_0_dev
flash_0_dev

Functions

int **board_pin_string_to_device_index** (const char **str_p*)

Convert given pin string to the pin number.

Return Pin number of negative error code.

Parameters

- *str_p*: Pin as a string.

arduino_mega — Arduino Mega

Source code: [src/boards/arduino_mega/board.h](#), [src/boards/arduino_mega/board.c](#)

Hardware reference: *Arduino Mega*

Defines

`pin_d0_dev`

`pin_d1_dev`

`pin_d2_dev`

`pin_d3_dev`

`pin_d4_dev`

`pin_d5_dev`

`pin_d6_dev`

`pin_d7_dev`

`pin_d8_dev`

`pin_d9_dev`

`pin_d10_dev`

`pin_d11_dev`

`pin_d12_dev`

`pin_d13_dev`

`pin_d14_dev`

`pin_d15_dev`

`pin_d16_dev`

`pin_d17_dev`

`pin_d18_dev`

`pin_d19_dev`

`pin_d20_dev`

`pin_d21_dev`

pin_d22_dev
pin_d23_dev
pin_d24_dev
pin_d25_dev
pin_d26_dev
pin_d27_dev
pin_d28_dev
pin_d29_dev
pin_d30_dev
pin_d31_dev
pin_d32_dev
pin_d33_dev
pin_d34_dev
pin_d35_dev
pin_d36_dev
pin_d37_dev
pin_d38_dev
pin_d39_dev
pin_d40_dev
pin_d41_dev
pin_d42_dev
pin_d43_dev
pin_d44_dev
pin_d45_dev
pin_d46_dev
pin_d47_dev
pin_d48_dev
pin_d49_dev
pin_d50_dev
pin_d51_dev
pin_d52_dev
pin_d53_dev
pin_a0_dev
pin_a1_dev
pin_a2_dev
pin_a3_dev

pin_a4_dev
pin_a5_dev
pin_a6_dev
pin_a7_dev
pin_a8_dev
pin_a9_dev
pin_a10_dev
pin_a11_dev
pin_a12_dev
pin_a13_dev
pin_a14_dev
pin_a15_dev
pin_led_dev
exti_d2_dev
exti_d3_dev
exti_d18_dev
exti_d19_dev
exti_d20_dev
exti_d21_dev
pwm_d2_dev
pwm_d3_dev
pwm_d5_dev
pwm_d6_dev
pwm_d7_dev
pwm_d8_dev
pwm_d9_dev
pwm_d10_dev
pwm_d11_dev
pwm_d12_dev
adc_0_dev
i2c_0_dev

Functions

int **board_pin_string_to_device_index** (const char **str_p*)
Convert given pin string to the pin number.

Return Pin number of negative error code.

Parameters

- `str_p`: Pin as a string.

arduino_nano — Arduino Nano

Source code: [src/boards/arduino_nano/board.h](#), [src/boards/arduino_nano/board.c](#)

Hardware reference: *Arduino Nano*

Defines

`pin_d2_dev`
`pin_d3_dev`
`pin_d4_dev`
`pin_d5_dev`
`pin_d6_dev`
`pin_d7_dev`
`pin_d8_dev`
`pin_d9_dev`
`pin_d10_dev`
`pin_d11_dev`
`pin_d12_dev`
`pin_d13_dev`
`pin_a0_dev`
`pin_a1_dev`
`pin_a2_dev`
`pin_a3_dev`
`pin_a4_dev`
`pin_a5_dev`
`pin_led_dev`
`exti_d2_dev`
`exti_d3_dev`
`pwm_d3_dev`
`pwm_d9_dev`
`pwm_d10_dev`
`pwm_d11_dev`
`adc_0_dev`
`i2c_0_dev`

Functions

int **board_pin_string_to_device_index** (const char **str_p*)

Convert given pin string to the pin number.

Return Pin number of negative error code.

Parameters

- *str_p*: Pin as a string.

arduino_pro_micro — Arduino Pro Micro

Source code: [src/boards/arduino_pro_micro/board.h](#), [src/boards/arduino_pro_micro/board.c](#)

Hardware reference: *Arduino Pro Micro*

Defines

`pin_d2_dev`

`pin_d3_dev`

`pin_d4_dev`

`pin_d5_dev`

`pin_d6_dev`

`pin_d7_dev`

`pin_d8_dev`

`pin_d9_dev`

`pin_d10_dev`

`pin_d14_dev`

`pin_d15_dev`

`pin_d16_dev`

`pin_a0_dev`

`pin_a1_dev`

`pin_a2_dev`

`pin_a3_dev`

`pin_led_dev`

`exti_d2_dev`

`exti_d3_dev`

`pwm_d3_dev`

`pwm_d9_dev`

`pwm_d10_dev`

`pwm_d11_dev`
`adc_0_dev`
`i2c_0_dev`

Functions

`int board_pin_string_to_device_index (const char *str_p)`
Convert given pin string to the pin number.

Return Pin number of negative error code.

Parameters

- `str_p`: Pin as a string.

`arduino_uno` — Arduino Uno

Source code: [src/boards/arduino_uno/board.h](#), [src/boards/arduino_uno/board.c](#)

Hardware reference: *Arduino Uno*

Defines

`pin_d2_dev`
`pin_d3_dev`
`pin_d4_dev`
`pin_d5_dev`
`pin_d6_dev`
`pin_d7_dev`
`pin_d8_dev`
`pin_d9_dev`
`pin_d10_dev`
`pin_d11_dev`
`pin_d12_dev`
`pin_d13_dev`
`pin_a0_dev`
`pin_a1_dev`
`pin_a2_dev`
`pin_a3_dev`
`pin_a4_dev`
`pin_a5_dev`

`pin_led_dev`
`exti_d2_dev`
`exti_d3_dev`
`pwm_d3_dev`
`pwm_d9_dev`
`pwm_d10_dev`
`pwm_d11_dev`
`adc_0_dev`
`i2c_0_dev`

Functions

`int board_pin_string_to_device_index (const char *str_p)`

Convert given pin string to the pin number.

Return Pin number of negative error code.

Parameters

- `str_p`: Pin as a string.

cygwin — Cygwin

Source code: [src/boards/cygwin/board.h](#), [src/boards/cygwin/board.c](#)

Defines

`PIN_DEVICE_BASE`
`pin_d2_dev`
`pin_d3_dev`
`pin_d4_dev`
`pin_d5_dev`
`pin_d6_dev`
`pin_d7_dev`
`pin_d8_dev`
`pin_d9_dev`
`pin_d10_dev`
`pin_d11_dev`
`pin_d12_dev`
`pin_d13_dev`

`pin_a0_dev`
`pin_a1_dev`
`pin_a2_dev`
`pin_a3_dev`
`pin_a4_dev`
`pin_a5_dev`
`pin_a6_dev`
`pin_a7_dev`
`pin_led_dev`
`pwm_d3_dev`
`pwm_d9_dev`
`pwm_d10_dev`
`pwm_d11_dev`
`adc_0_dev`

Functions

`int board_pin_string_to_device_index (const char *str_p)`
Convert given pin string to the pin number.

Return Pin number of negative error code.

Parameters

- `str_p`: Pin as a string.

esp01 — ESP8266 Development Board

Source code: [src/boards/esp01/board.h](#), [src/boards/esp01/board.c](#)

Hardware reference: [ESP-01](#)

Defines

`pin_gpio0_dev`
`pin_gpio1_dev`
`pin_gpio2_dev`
`pin_d0_dev`
`pin_d1_dev`
`pin_d2_dev`
`pin_led_dev`
`flash_0_dev`

Functions

int **board_pin_string_to_device_index** (const char **str_p*)

Convert given pin string to the pin number.

Return Pin number of negative error code.

Parameters

- *str_p*: Pin as a string.

esp12e — ESP8266 Development Board

Source code: [src/boards/esp12e/board.h](#), [src/boards/esp12e/board.c](#)

Hardware reference: *ESP-12E Development Board*

Defines

`pin_gpio0_dev`
`pin_gpio2_dev`
`pin_gpio4_dev`
`pin_gpio5_dev`
`pin_gpio12_dev`
`pin_gpio13_dev`
`pin_gpio14_dev`
`pin_gpio15_dev`
`pin_d0_dev`
`pin_d2_dev`
`pin_d4_dev`
`pin_d5_dev`
`pin_d12_dev`
`pin_d13_dev`
`pin_d14_dev`
`pin_d15_dev`
`pin_led_dev`
`pin_a0_dev`
`adc_0_dev`
`flash_0_dev`

Functions

int **board_pin_string_to_device_index** (const char **str_p*)

Convert given pin string to the pin number.

Return Pin number of negative error code.

Parameters

- *str_p*: Pin as a string.

linux — Linux

Source code: [src/boards/linux/board.h](#), [src/boards/linux/board.c](#)

Defines

PIN_DEVICE_BASE

pin_d2_dev

pin_d3_dev

pin_d4_dev

pin_d5_dev

pin_d6_dev

pin_d7_dev

pin_d8_dev

pin_d9_dev

pin_d10_dev

pin_d11_dev

pin_d12_dev

pin_d13_dev

pin_a0_dev

pin_a1_dev

pin_a2_dev

pin_a3_dev

pin_a4_dev

pin_a5_dev

pin_a6_dev

pin_a7_dev

pin_led_dev

pwm_d3_dev

`pwm_d9_dev`
`pwm_d10_dev`
`pwm_d11_dev`
`adc_0_dev`
`pin_dac0_dev`
`pin_dac1_dev`

Functions

`int board_pin_string_to_device_index (const char *str_p)`
Convert given pin string to the pin number.

Return Pin number of negative error code.

Parameters

- `str_p`: Pin as a string.

`nano32` — **Nano32**

Source code: [src/boards/nano32/board.h](#), [src/boards/nano32/board.c](#)

Hardware reference: *Nano32*

Defines

`pin_gpio00_dev`
`pin_gpio01_dev`
`pin_gpio02_dev`
`pin_gpio03_dev`
`pin_gpio04_dev`
`pin_gpio05_dev`
`pin_gpio06_dev`
`pin_gpio07_dev`
`pin_gpio08_dev`
`pin_gpio09_dev`
`pin_gpio10_dev`
`pin_gpio11_dev`
`pin_gpio12_dev`
`pin_gpio13_dev`
`pin_gpio14_dev`

pin_gpio15_dev
pin_gpio16_dev
pin_gpio17_dev
pin_gpio18_dev
pin_gpio19_dev
pin_gpio21_dev
pin_gpio22_dev
pin_gpio23_dev
pin_gpio25_dev
pin_gpio26_dev
pin_gpio27_dev
pin_gpio32_dev
pin_gpio33_dev
pin_gpio34_dev
pin_gpio35_dev
pin_gpio36_dev
pin_gpio39_dev
pin_led_dev
pin_adc0_dev
pin_adc3_dev
pin_adc4_dev
pin_adc5_dev
pin_adc6_dev
pin_adc7_dev
pin_adc10_dev
pin_adc11_dev
pin_adc12_dev
pin_adc13_dev
pin_adc14_dev
pin_adc15_dev
pin_adc16_dev
pin_adc17_dev
pin_adc18_dev
pin_adc19_dev
pin_dac1_dev
pin_dac2_dev

`i2c_dev`
`spi_h_dev`
`spi_v_dev`
`adc_1_dev`
`adc_2_dev`
`flash_0_dev`

Functions

int **board_pin_string_to_device_index** (const char **str_p*)
Convert given pin string to the pin number.

Return Pin number of negative error code.

Parameters

- `str_p`: Pin as a string.

photon — Photon

Source code: [src/boards/photon/board.h](#), [src/boards/photon/board.c](#)

Hardware reference: *Photon*

Defines

`pin_d0_dev`
`pin_d1_dev`
`pin_d2_dev`
`pin_d3_dev`
`pin_d4_dev`
`pin_d5_dev`
`pin_d6_dev`
`pin_d7_dev`
`pin_a0_dev`
`pin_a1_dev`
`pin_a2_dev`
`pin_a3_dev`
`pin_a4_dev`
`pin_a5_dev`
`pin_led_dev`

`pin_dac0_dev`
`pin_dac1_dev`
`pwm_d0_dev`
`pwm_d1_dev`
`pwm_d2_dev`
`pwm_d3_dev`
`pwm_a4_dev`
`pwm_a5_dev`
`flash_0_dev`
`sdio_0_dev`

Functions

`int board_pin_string_to_device_index (const char *str_p)`

Convert given pin string to the pin number.

Return Pin number of negative error code.

Parameters

- `str_p`: Pin as a string.

stm32f3discovery — STM32F3DISCOVERY

Source code: [src/boards/stm32f3discovery/board.h](#), [src/boards/stm32f3discovery/board.c](#)

Hardware reference: [STM32F3DISCOVERY](#)

Defines

`pin_pa0_dev`
`pin_pa1_dev`
`pin_pa2_dev`
`pin_pa3_dev`
`pin_pa4_dev`
`pin_pa5_dev`
`pin_pa6_dev`
`pin_pa7_dev`
`pin_pa8_dev`
`pin_pa9_dev`
`pin_pa10_dev`

pin_pa11_dev
pin_pa12_dev
pin_pa13_dev
pin_pa14_dev
pin_pa15_dev
pin_pb0_dev
pin_pb1_dev
pin_pb2_dev
pin_pb3_dev
pin_pb4_dev
pin_pb5_dev
pin_pb6_dev
pin_pb7_dev
pin_pb8_dev
pin_pb9_dev
pin_pb10_dev
pin_pb11_dev
pin_pb12_dev
pin_pb13_dev
pin_pb14_dev
pin_pb15_dev
pin_pc0_dev
pin_pc1_dev
pin_pc2_dev
pin_pc3_dev
pin_pc4_dev
pin_pc5_dev
pin_pc6_dev
pin_pc7_dev
pin_pc8_dev
pin_pc9_dev
pin_pc10_dev
pin_pc11_dev
pin_pc12_dev
pin_pc13_dev
pin_pc14_dev

pin_pc15_dev
pin_pd0_dev
pin_pd1_dev
pin_pd2_dev
pin_pd3_dev
pin_pd4_dev
pin_pd5_dev
pin_pd6_dev
pin_pd7_dev
pin_pd8_dev
pin_pd9_dev
pin_pd10_dev
pin_pd11_dev
pin_pd12_dev
pin_pd13_dev
pin_pd14_dev
pin_pd15_dev
pin_pe0_dev
pin_pe1_dev
pin_pe2_dev
pin_pe3_dev
pin_pe4_dev
pin_pe5_dev
pin_pe6_dev
pin_pe7_dev
pin_pe8_dev
pin_pe9_dev
pin_pe10_dev
pin_pe11_dev
pin_pe12_dev
pin_pe13_dev
pin_pe14_dev
pin_pe15_dev
uart_0_dev
uart_1_dev
uart_2_dev

`spi_0_dev`
`spi_1_dev`
`spi_2_dev`
`i2c_0_dev`
`i2c_1_dev`
`can_0_dev`
`flash_0_dev`

Functions

`int board_pin_string_to_device_index (const char *str_p)`
Convert given pin string to the pin number.

Return Pin number of negative error code.

Parameters

- `str_p`: Pin as a string.

`stm32vldiscovery` — **STM32VLDISCOVERY**

Source code: [src/boards/stm32vldiscovery/board.h](#), [src/boards/stm32vldiscovery/board.c](#)

Hardware reference: *STM32VLDISCOVERY*

Defines

`pin_pa0_dev`
`pin_pa1_dev`
`pin_pa2_dev`
`pin_pa3_dev`
`pin_pa4_dev`
`pin_pa5_dev`
`pin_pa6_dev`
`pin_pa7_dev`
`pin_pa8_dev`
`pin_pa9_dev`
`pin_pa10_dev`
`pin_pa11_dev`
`pin_pa12_dev`
`pin_pa13_dev`

pin_pa14_dev
pin_pa15_dev
pin_pb0_dev
pin_pb1_dev
pin_pb2_dev
pin_pb3_dev
pin_pb4_dev
pin_pb5_dev
pin_pb6_dev
pin_pb7_dev
pin_pb8_dev
pin_pb9_dev
pin_pb10_dev
pin_pb11_dev
pin_pb12_dev
pin_pb13_dev
pin_pb14_dev
pin_pb15_dev
pin_pc0_dev
pin_pc1_dev
pin_pc2_dev
pin_pc3_dev
pin_pc4_dev
pin_pc5_dev
pin_pc6_dev
pin_pc7_dev
pin_pc8_dev
pin_pc9_dev
pin_pc10_dev
pin_pc11_dev
pin_pc12_dev
pin_pc13_dev
pin_pc14_dev
pin_pc15_dev
pin_pd0_dev
pin_pd1_dev

`pin_pd2_dev`
`pin_led_dev`
`pin_ld3_dev`
`pin_ld4_dev`
`uart_0_dev`
`uart_1_dev`
`uart_2_dev`
`spi_0_dev`
`spi_1_dev`
`spi_2_dev`
`i2c_0_dev`
`i2c_1_dev`
`flash_0_dev`

Functions

`int board_pin_string_to_device_index (const char *str_p)`

Convert given pin string to the pin number.

Return Pin number of negative error code.

Parameters

- `str_p`: Pin as a string.

1.6.15 mcus

The Micro Controller Units (MCU:s) supported by *Simba*.

The MCU:s on [Github](#).

atmega2560 — ATmega2560

Source code: [src/mcus/atmega2560/mcu.h](#)

Defines

`PIN_DEVICE_MAX`
`EXTI_DEVICE_MAX`
`SPI_DEVICE_MAX`
`UART_DEVICE_MAX`
`PWM_DEVICE_MAX`

`ADC_DEVICE_MAX`

`I2C_DEVICE_MAX`

atmega328p — ATmega328p

Source code: [src/mcus/atmega328p/mcu.h](#)

Defines

`PIN_DEVICE_MAX`

`EXTI_DEVICE_MAX`

`SPI_DEVICE_MAX`

`UART_DEVICE_MAX`

`PWM_DEVICE_MAX`

`ADC_DEVICE_MAX`

`I2C_DEVICE_MAX`

`USART0_TX_vect`

`USART0_RX_vect`

`USART0_UDRE_vect`

atmega32u4 — ATmega32u4

Source code: [src/mcus/atmega32u4/mcu.h](#)

Defines

`PIN_DEVICE_MAX`

`EXTI_DEVICE_MAX`

`SPI_DEVICE_MAX`

`UART_DEVICE_MAX`

`PWM_DEVICE_MAX`

`ADC_DEVICE_MAX`

`I2C_DEVICE_MAX`

`USB_DEVICE_MAX`

`USART0_TX_vect`

`USART0_RX_vect`

`USART0_UDRE_vect`

UCSZ00
UCSZ01
UCSZ02
UPM00
UPM01
USBS0
U2X0
UPE0
DOR0
FE0
TXC0
RXCIE0
RXEN0
TXEN0
UDRE0
UDRIE0
TXCIE0

esp32 — Esp32

Hardware reference: <https://github.com/eerimoq/hardware-reference/tree/master/esp32>

Source code: `src/mcus/esp32/mcu.h`

Defines

PIN_DEVICE_MAX
EXTI_DEVICE_MAX
SPI_DEVICE_MAX
UART_DEVICE_MAX
ADC_DEVICE_MAX
I2C_DEVICE_MAX
FLASH_DEVICE_MAX
CAN_DEVICE_MAX

esp8266 — Esp8266

Hardware reference: <https://github.com/erimoq/hardware-reference/tree/master/esp8266>

Source code: [src/mcus/esp8266/mcu.h](#)

Defines

`PIN_DEVICE_MAX`
`EXTI_DEVICE_MAX`
`SPI_DEVICE_MAX`
`UART_DEVICE_MAX`
`PWM_DEVICE_MAX`
`ADC_DEVICE_MAX`
`FLASH_DEVICE_MAX`

linux — Linux

Source code: [src/mcus/linux/mcu.h](#)

Defines

`PIN_DEVICE_MAX`
`EXTI_DEVICE_MAX`
`SPI_DEVICE_MAX`
`UART_DEVICE_MAX`
`CAN_DEVICE_MAX`
`PWM_DEVICE_MAX`
`ADC_DEVICE_MAX`
`FLASH_DEVICE_MAX`
`DAC_DEVICE_MAX`

sam3x8e — SAM3X8E

Source code: [src/mcus/sam/mcu.h](#)

Defines

SAM_PA

SAM_PB

SAM_PC

SAM_PD

stm32f100rb — STM32F100RB

Source code: [src/mcus/stm32f100rb/mcu.h](#)

Defines

PIN_DEVICE_MAX

UART_DEVICE_MAX

SPI_DEVICE_MAX

I2C_DEVICE_MAX

CAN_DEVICE_MAX

FLASH_DEVICE_MAX

stm32f205rg — STM32F205RG

Source code: [src/mcus/stm32f205rg/mcu.h](#)

Defines

PIN_DEVICE_MAX

UART_DEVICE_MAX

SPI_DEVICE_MAX

I2C_DEVICE_MAX

CAN_DEVICE_MAX

FLASH_DEVICE_MAX

SDIO_DEVICE_MAX

stm32f303vc — STM32F303VC

Source code: [src/mcus/stm32f303vc/mcu.h](#)

Defines

`PIN_DEVICE_MAX`

`UART_DEVICE_MAX`

`SPI_DEVICE_MAX`

`I2C_DEVICE_MAX`

`CAN_DEVICE_MAX`

`FLASH_DEVICE_MAX`

1.7 License

The MIT License (MIT)

Copyright (c) 2014-2016, Erik Moqvist

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.

1.8 Videos

1.8.1 #6 Simba: CAN client-server test suite on Nano32 (ESP32) and Arduino Due.

Transmit CAN frames between a Nano32 and an Arduino Due.

1.8.2 #5 Simba: Room temperature (DS18B20).

Read and print the room temperature measured with a DS18B20 sensor.

1.8.3 #4 Simba: Hello world.

This application prints “Hello world!” to standard output.

1.8.4 #3 Simba: Analog read.

Read the value of an analog pin periodically once every second and print the read value to standard output.

1.8.5 #2 Simba: Blink example.

This video demonstrates the classic blink application. It's run on a Arduino Due that has a SAM2X8E ARM MCU.

1.8.6 #1 Simba: Gource of the Simba repository.

Gource visualizes the Simba Git repository file tree over time. In this project the source, test and documentation was written simultaneously, a perfect school book example of software development.

1.9 Links

This page contains links to external websites that are related to Simba.

Feel free to add your project to the list by submitting a pull request of [this page](#) on Github.

1.9.1 Pumbaa - MicroPython on Simba

Python on microcontrollers thanks to MicroPython (and in this case Simba).

Documentation: <http://pumbaa.readthedocs.io>

Github: <https://github.com/eerimoq/pumbaa>

MicroPython: <http://www.micropython.org>

1.9.2 Wingfence

A BWF for a home made robot mower.

Github: <https://github.com/wingstar74/wingfence>

Features

- *Threads* scheduled by a priority based cooperative or preemptive scheduler.
- Channels for inter-thread communication (*Queue*, *Event*).
- *Timers*.
- *Counting semaphores*.
- Device drivers (*SPI*, *UART*, ...)
- A simple *shell*.
- *Logging*.
- Internet protocols (*TCP*, *UDP*, *HTTP*, ...).
- *Debug file system*.
- File systems (*FAT16*, *SPIFFS*).

See the *Library Reference* for a full list of features.

Testing

To ensure high code quality each module is tested extensively by many test suites. See [Testing](#) for details.

Design goals

- Rapid development.
- Clean interfaces.
- Small memory footprint.
- No dynamic memory allocation.
- Portability.

Indices and tables

- `genindex`
- `modindex`
- `search`

a

adc, 122
analog_input_pin, 124
analog_output_pin, 125
arduino_due, 310
arduino_mega, 315
arduino_nano, 318
arduino_pro_micro, 319
arduino_uno, 320
atmega2560, 333
atmega328p, 334
atmega32u4, 334

b

base64, 300
bcm43362, 126
binary_tree, 281
bits, 282
bus, 187

c

can, 128
chan, 190
chipid, 130
circular_heap, 288
color, 291
configfile, 292
console, 264
crc, 305
cygwin, 321

d

dac, 130
ds18b20, 131
ds3231, 133

e

errno, 102
esp01, 322
esp12e, 323

esp32, 335
esp8266, 336
esp_wifi, 134
esp_wifi_softap, 134
esp_wifi_station, 136
event, 195
exti, 139

f

fat16, 203
fifo, 283
flash, 140
fs, 215

h

harness, 274
hash_map, 284
heap, 290
http_server, 240
http_websocket_client, 244
http_websocket_server, 245

i

i2c, 142
i2c_soft, 144
inet, 246

j

json, 300

l

linux, 336
list, 286
log, 276

m

mcp2515, 146
midi, 307
mqtt_client, 248

n

nano32, [325](#)
network_interface, [251](#)
network_interface_driver_esp, [253](#)
network_interface_slip, [252](#)
network_interface_wifi, [253](#)
nrf24l01, [148](#)

o

owi, [150](#)

p

photon, [327](#)
pin, [152](#)
ping, [258](#)
pwm, [154](#)

q

queue, [196](#)

r

re, [294](#)
rwlock, [200](#)

s

sam3x8e, [336](#)
sd, [155](#)
sdio, [160](#)
sem, [202](#)
service, [265](#)
settings, [267](#)
sha1, [305](#)
shell, [271](#)
socket, [259](#)
spi, [163](#)
spiffs, [227](#)
std, [296](#)
stm32f100rb, [337](#)
stm32f205rg, [337](#)
stm32f303vc, [337](#)
stm32f3discovery, [328](#)
stm32vldiscovery, [331](#)
sys, [108](#)

t

thrd, [111](#)
time, [118](#)
timer, [119](#)
types, [121](#)

u

uart, [166](#)
uart_soft, [168](#)

usb, [169](#)
usb_device, [177](#)
usb_device_class_cdc, [177](#)
usb_host, [180](#)
usb_host_class_hid, [181](#)
usb_host_class_mass_storage, [182](#)

w

watchdog, [187](#)

Symbols

`_ASSERTFMT` (C macro), 122

A

`adc` (module), 122
`adc_0_dev` (C macro), 314, 317, 318, 320–323, 325
`adc_1_dev` (C macro), 327
`adc_2_dev` (C macro), 327
`adc_async_convert` (C++ function), 123
`adc_async_wait` (C++ function), 123
`adc_convert` (C++ function), 123
`adc_convert_isr` (C++ function), 124
`adc_device` (C++ member), 124
`ADC_DEVICE_MAX` (C macro), 333–336
`adc_init` (C++ function), 123
`adc_module_init` (C++ function), 123
`ADC_REFERENCE_VCC` (C macro), 122
`analog_input_pin` (module), 124
`analog_input_pin_init` (C++ function), 124
`analog_input_pin_module_init` (C++ function), 124
`analog_input_pin_read` (C++ function), 124
`analog_input_pin_read_isr` (C++ function), 125
`analog_input_pin_t` (C++ class), 125
`analog_input_pin_t::adc` (C++ member), 125
`analog_output_pin` (module), 125
`analog_output_pin_init` (C++ function), 125
`analog_output_pin_module_init` (C++ function), 125
`analog_output_pin_read` (C++ function), 126
`analog_output_pin_t` (C++ class), 126
`analog_output_pin_t::pwm` (C++ member), 126
`analog_output_pin_write` (C++ function), 125
`arduino_due` (module), 310
`arduino_mega` (module), 315
`arduino_nano` (module), 318
`arduino_pro_micro` (module), 319
`arduino_uno` (module), 320
`ASSERT` (C macro), 122
`ASSERTN` (C macro), 122
`atmega2560` (module), 333
`atmega328p` (module), 334

`atmega32u4` (module), 334

B

`base64` (module), 300
`base64_decode` (C++ function), 300
`base64_encode` (C++ function), 300
`bcm43362` (module), 126
`bcm43362_connect` (C++ function), 127
`bcm43362_disconnect` (C++ function), 127
`bcm43362_driver_t` (C++ class), 128
`bcm43362_driver_t::sdio` (C++ member), 128
`bcm43362_init` (C++ function), 126
`bcm43362_module_init` (C++ function), 126
`bcm43362_read` (C++ function), 127
`bcm43362_start` (C++ function), 126
`bcm43362_stop` (C++ function), 127
`bcm43362_write` (C++ function), 127
`binary_tree` (module), 281
`binary_tree_delete` (C++ function), 281
`binary_tree_init` (C++ function), 281
`binary_tree_insert` (C++ function), 281
`binary_tree_node_t` (C++ class), 282
`binary_tree_node_t::height` (C++ member), 282
`binary_tree_node_t::key` (C++ member), 282
`binary_tree_node_t::left_p` (C++ member), 282
`binary_tree_node_t::right_p` (C++ member), 282
`binary_tree_print` (C++ function), 282
`binary_tree_search` (C++ function), 282
`binary_tree_t` (C++ class), 282
`binary_tree_t::root_p` (C++ member), 282
`BIT` (C macro), 122
`BITFIELD_GET` (C macro), 122
`BITFIELD_SET` (C macro), 122
`bits` (module), 282
`bits_insert_32` (C++ function), 282
`board_pin_string_to_device_index` (C++ function), 315, 317, 319–325, 327, 328, 331, 333
`bpb_t` (C++ class), 210
`bpb_t::bytes_per_sector` (C++ member), 210
`bpb_t::fat_count` (C++ member), 210
`bpb_t::head_count` (C++ member), 211

bpb_t::hidden_sectors (C++ member), 211
 bpb_t::media_type (C++ member), 211
 bpb_t::reserved_sector_count (C++ member), 210
 bpb_t::root_dir_entry_count (C++ member), 211
 bpb_t::sectors_per_cluster (C++ member), 210
 bpb_t::sectors_per_fat (C++ member), 211
 bpb_t::sectors_per_track (C++ member), 211
 bpb_t::total_sectors_large (C++ member), 211
 bpb_t::total_sectors_small (C++ member), 211
 BTASSERT (C macro), 275
 BTASSERTN (C macro), 275
 bus (module), 187
 bus_attach (C++ function), 188
 bus_detach (C++ function), 189
 bus_init (C++ function), 188
 bus_listener_init (C++ function), 188
 bus_listener_t (C++ class), 189
 bus_listener_t::base (C++ member), 189
 bus_listener_t::chan_p (C++ member), 189
 bus_listener_t::id (C++ member), 189
 bus_listener_t::next_p (C++ member), 189
 bus_module_init (C++ function), 188
 bus_t (C++ class), 189
 bus_t::listeners (C++ member), 189
 bus_t::rwlock (C++ member), 189
 bus_write (C++ function), 189

C

can (module), 128
 can_0_dev (C macro), 331
 can_device (C++ member), 130
 CAN_DEVICE_MAX (C macro), 335–338
 can_frame_t (C++ class), 130
 can_frame_t::extended_frame (C++ member), 130
 can_frame_t::id (C++ member), 130
 can_frame_t::rtr (C++ member), 130
 can_frame_t::size (C++ member), 130
 can_frame_t::timestamp (C++ member), 130
 can_frame_t::u32 (C++ member), 130
 can_frame_t::u8 (C++ member), 130
 can_init (C++ function), 128
 can_read (C++ function), 129
 CAN_SPEED_1000KBPS (C macro), 128
 CAN_SPEED_250KBPS (C macro), 128
 CAN_SPEED_500KBPS (C macro), 128
 can_start (C++ function), 129
 can_stop (C++ function), 129
 can_write (C++ function), 129
 chan (module), 190
 chan_init (C++ function), 191
 chan_is_polled_isr (C++ function), 193
 chan_list_add (C++ function), 193
 chan_list_destroy (C++ function), 193
 chan_list_init (C++ function), 193

chan_list_poll (C++ function), 193
 chan_list_remove (C++ function), 193
 chan_list_t (C++ class), 194
 chan_list_t::chans_pp (C++ member), 194
 chan_list_t::flags (C++ member), 194
 chan_list_t::len (C++ member), 194
 chan_list_t::max (C++ member), 194
 chan_module_init (C++ function), 191
 chan_null (C++ function), 194
 chan_poll (C++ function), 194
 chan_read (C++ function), 192
 chan_read_fn_t (C++ type), 190
 chan_read_null (C++ function), 194
 chan_set_write_filter_cb (C++ function), 191
 chan_set_write_filter_isr_cb (C++ function), 191
 chan_set_write_isr_cb (C++ function), 191
 chan_size (C++ function), 192
 chan_size_fn_t (C++ type), 190
 chan_size_null (C++ function), 194
 chan_t (C++ class), 194
 chan_t::list_p (C++ member), 195
 chan_t::read (C++ member), 195
 chan_t::reader_p (C++ member), 195
 chan_t::size (C++ member), 195
 chan_t::write (C++ member), 195
 chan_t::write_filter_cb (C++ member), 195
 chan_t::write_filter_isr_cb (C++ member), 195
 chan_t::write_isr (C++ member), 195
 chan_t::writer_p (C++ member), 195
 chan_write (C++ function), 192
 chan_write_filter_fn_t (C++ type), 190
 chan_write_fn_t (C++ type), 190
 chan_write_isr (C++ function), 192
 chan_write_null (C++ function), 194
 chipid (module), 130
 chipid_read (C++ function), 130
 circular_heap (module), 288
 circular_heap_alloc (C++ function), 289
 circular_heap_free (C++ function), 289
 circular_heap_init (C++ function), 289
 circular_heap_t (C++ class), 289
 circular_heap_t::alloc_p (C++ member), 289
 circular_heap_t::begin_p (C++ member), 289
 circular_heap_t::end_p (C++ member), 289
 circular_heap_t::free_p (C++ member), 289
 COLOR (C macro), 292
 color (module), 291
 COLOR_BACKGROUND_BLACK (C macro), 292
 COLOR_BACKGROUND_BLUE (C macro), 292
 COLOR_BACKGROUND_CYAN (C macro), 292
 COLOR_BACKGROUND_DEFAULT (C macro), 292
 COLOR_BACKGROUND_GREEN (C macro), 292
 COLOR_BACKGROUND_MAGENTA (C macro), 292
 COLOR_BACKGROUND_RED (C macro), 292

COLOR_BACKGROUND_WHITE (C macro), 292
 COLOR_BACKGROUND_YELLOW (C macro), 292
 COLOR_BOLD_OFF (C macro), 292
 COLOR_BOLD_ON (C macro), 291
 COLOR_FOREGROUND_BLACK (C macro), 292
 COLOR_FOREGROUND_BLUE (C macro), 292
 COLOR_FOREGROUND_CYAN (C macro), 292
 COLOR_FOREGROUND_DEFAULT (C macro), 292
 COLOR_FOREGROUND_GREEN (C macro), 292
 COLOR_FOREGROUND_MAGENTA (C macro), 292
 COLOR_FOREGROUND_RED (C macro), 292
 COLOR_FOREGROUND_WHITE (C macro), 292
 COLOR_FOREGROUND_YELLOW (C macro), 292
 COLOR_INVERSE_OFF (C macro), 292
 COLOR_INVERSE_ON (C macro), 291
 COLOR_ITALICS_OFF (C macro), 292
 COLOR_ITALICS_ON (C macro), 291
 COLOR_RESET (C macro), 291
 COLOR_STRIKETHROUGH_OFF (C macro), 292
 COLOR_STRIKETHROUGH_ON (C macro), 291
 COLOR_UNDERLINE_OFF (C macro), 292
 COLOR_UNDERLINE_ON (C macro), 291
 CONFIG_ASSERT (C macro), 11
 CONFIG_DEBUG (C macro), 11
 CONFIG_FS_CMD_DS18B20_LIST (C macro), 11
 CONFIG_FS_CMD_ESP_WIFI_STATUS (C macro), 11
 CONFIG_FS_CMD_FS_APPEND (C macro), 11
 CONFIG_FS_CMD_FS_COUNTERS_LIST (C macro), 11
 CONFIG_FS_CMD_FS_COUNTERS_RESET (C macro), 11
 CONFIG_FS_CMD_FS_FILESYSTEMS_LIST (C macro), 11
 CONFIG_FS_CMD_FS_FORMAT (C macro), 11
 CONFIG_FS_CMD_FS_LIST (C macro), 11
 CONFIG_FS_CMD_FS_PARAMETERS_LIST (C macro), 12
 CONFIG_FS_CMD_FS_READ (C macro), 12
 CONFIG_FS_CMD_FS_REMOVE (C macro), 12
 CONFIG_FS_CMD_FS_WRITE (C macro), 12
 CONFIG_FS_CMD_I2C_READ (C macro), 12
 CONFIG_FS_CMD_I2C_WRITE (C macro), 12
 CONFIG_FS_CMD_LOG_LIST (C macro), 12
 CONFIG_FS_CMD_LOG_PRINT (C macro), 12
 CONFIG_FS_CMD_LOG_SET_LOG_MASK (C macro), 12
 CONFIG_FS_CMD_NETWORK_INTERFACE_LIST (C macro), 12
 CONFIG_FS_CMD_PIN_READ (C macro), 12
 CONFIG_FS_CMD_PIN_SET_MODE (C macro), 12
 CONFIG_FS_CMD_PIN_WRITE (C macro), 12
 CONFIG_FS_CMD_PING_PING (C macro), 12
 CONFIG_FS_CMD_SERVICE_LIST (C macro), 12
 CONFIG_FS_CMD_SERVICE_START (C macro), 12
 CONFIG_FS_CMD_SERVICE_STOP (C macro), 12
 CONFIG_FS_CMD_SETTINGS_LIST (C macro), 12
 CONFIG_FS_CMD_SETTINGS_READ (C macro), 12
 CONFIG_FS_CMD_SETTINGS_RESET (C macro), 12
 CONFIG_FS_CMD_SETTINGS_WRITE (C macro), 12
 CONFIG_FS_CMD_SYS_CONFIG (C macro), 13
 CONFIG_FS_CMD_SYS_INFO (C macro), 13
 CONFIG_FS_CMD_SYS_REBOOT (C macro), 13
 CONFIG_FS_CMD_SYS_UPTIME (C macro), 13
 CONFIG_FS_CMD_THRD_LIST (C macro), 13
 CONFIG_FS_CMD_THRD_SET_LOG_MASK (C macro), 13
 CONFIG_FS_CMD_USB_DEVICE_LIST (C macro), 13
 CONFIG_FS_CMD_USB_HOST_LIST (C macro), 13
 CONFIG_FS_PATH_MAX (C macro), 13
 CONFIG_MONITOR_THREAD (C macro), 13
 CONFIG_PREEMPTIVE_SCHEDULER (C macro), 13
 CONFIG_PROFILE_STACK (C macro), 13
 CONFIG_SETTINGS_AREA_SIZE (C macro), 13
 CONFIG_SHELL_COMMAND_MAX (C macro), 13
 CONFIG_SHELL_HISTORY_SIZE (C macro), 13
 CONFIG_SHELL_MINIMAL (C macro), 13
 CONFIG_SHELL_PROMPT (C macro), 13
 CONFIG_SOCKET_RAW (C macro), 13
 CONFIG_SPIFFS (C macro), 13
 CONFIG_START_CONSOLE (C macro), 13
 CONFIG_START_CONSOLE_DEVICE_INDEX (C macro), 13
 CONFIG_START_CONSOLE_UART_BAUDRATE (C macro), 14
 CONFIG_START_CONSOLE_USB_CDC_CONTROL_INTERFACE (C macro), 14
 CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_IN (C macro), 14
 CONFIG_START_CONSOLE_USB_CDC_ENDPOINT_OUT (C macro), 14
 CONFIG_START_CONSOLE_USB_CDC_WAIT_FOR_CONNECTION (C macro), 14
 CONFIG_START_FILESYSTEM (C macro), 14
 CONFIG_START_FILESYSTEM_ADDRESS (C macro), 14
 CONFIG_START_FILESYSTEM_SIZE (C macro), 14
 CONFIG_START_NETWORK (C macro), 14
 CONFIG_START_NETWORK_INTERFACE_WIFI_CONNECT_TIMEOUT (C macro), 14
 CONFIG_START_NETWORK_INTERFACE_WIFI_PASSWORD (C macro), 14
 CONFIG_START_NETWORK_INTERFACE_WIFI_SSID (C macro), 14
 CONFIG_START_SHELL (C macro), 14
 CONFIG_START_SHELL_PRIO (C macro), 14
 CONFIG_START_SHELL_STACK_SIZE (C macro), 14

CONFIG_STD_OUTPUT_BUFFER_MAX (C macro), 14
 CONFIG_SYS_CONFIG_STRING (C macro), 11
 CONFIG_SYS_SIMBA_MAIN_STACK_MAX (C macro), 11
 CONFIG_SYSTEM_TICK_FREQUENCY (C macro), 14
 CONFIG_THRD_CPU_USAGE (C macro), 14
 CONFIG_THRD_ENV (C macro), 14
 CONFIG_THRD_IDLE_STACK_SIZE (C macro), 14
 CONFIG_THRD_SCHEDULED (C macro), 14
 CONFIG_THRD_TERMINATE (C macro), 15
 CONFIG_USB_DEVICE_PID (C macro), 15
 CONFIG_USB_DEVICE_VID (C macro), 15
 configfile (module), 292
 configfile_get (C++ function), 293
 configfile_get_float (C++ function), 294
 configfile_get_long (C++ function), 294
 configfile_init (C++ function), 293
 configfile_set (C++ function), 293
 configfile_t (C++ class), 294
 configfile_t::buf_p (C++ member), 294
 configfile_t::size (C++ member), 294
 CONFIGURATION_ATTRIBUTES_BUS_POWERED (C macro), 171
 console (module), 264
 console_get_input_channel (C++ function), 265
 console_get_output_channel (C++ function), 265
 console_init (C++ function), 264
 console_module_init (C++ function), 264
 console_set_input_channel (C++ function), 264
 console_set_output_channel (C++ function), 265
 console_start (C++ function), 264
 console_stop (C++ function), 264
 container_of (C macro), 122
 crc (module), 305
 crc_32 (C++ function), 305
 crc_7 (C++ function), 305
 crc_ccitt (C++ function), 305
 crc_xmodem (C++ function), 305
 cygwin (module), 321
D
 dac (module), 130
 dac_0_dev (C macro), 314
 dac_async_convert (C++ function), 131
 dac_async_wait (C++ function), 131
 dac_convert (C++ function), 131
 dac_device (C++ member), 131
 DAC_DEVICE_MAX (C macro), 336
 dac_init (C++ function), 130
 dac_module_init (C++ function), 130
 date_t (C++ class), 119
 date_t::date (C++ member), 119
 date_t::day (C++ member), 119
 date_t::hour (C++ member), 119
 date_t::minute (C++ member), 119
 date_t::month (C++ member), 119
 date_t::second (C++ member), 119
 date_t::year (C++ member), 119
 DESCRIPTOR_TYPE_CDC (C macro), 170
 DESCRIPTOR_TYPE_CONFIGURATION (C macro), 170
 DESCRIPTOR_TYPE_DEVICE (C macro), 169
 DESCRIPTOR_TYPE_ENDPOINT (C macro), 170
 DESCRIPTOR_TYPE_INTERFACE (C macro), 170
 DESCRIPTOR_TYPE_INTERFACE_ASSOCIATION (C macro), 170
 DESCRIPTOR_TYPE_RPIPE (C macro), 170
 DESCRIPTOR_TYPE_STRING (C macro), 170
 DIR_ATTR_ARCHIVE (C macro), 205
 DIR_ATTR_DIRECTORY (C macro), 205
 DIR_ATTR_HIDDEN (C macro), 205
 DIR_ATTR_READ_ONLY (C macro), 205
 DIR_ATTR_SYSTEM (C macro), 205
 DIR_ATTR_VOLUME_ID (C macro), 205
 dir_t (C++ class), 212
 dir_t::attributes (C++ member), 212
 dir_t::creation_date (C++ member), 213
 dir_t::creation_time (C++ member), 212
 dir_t::creation_time_tenths (C++ member), 212
 dir_t::file_size (C++ member), 213
 dir_t::first_cluster_high (C++ member), 213
 dir_t::first_cluster_low (C++ member), 213
 dir_t::last_access_date (C++ member), 213
 dir_t::last_write_date (C++ member), 213
 dir_t::last_write_time (C++ member), 213
 dir_t::name (C++ member), 212
 dir_t::reserved1 (C++ member), 212
 DIV_CEIL (C macro), 122
 DOR0 (C macro), 335
 ds18b20 (module), 131
 ds18b20_convert (C++ function), 132
 ds18b20_driver_t (C++ class), 133
 ds18b20_driver_t::next_p (C++ member), 133
 ds18b20_driver_t::owi_p (C++ member), 133
 DS18B20_FAMILY_CODE (C macro), 132
 ds18b20_get_temperature (C++ function), 132
 ds18b20_get_temperature_str (C++ function), 132
 ds18b20_init (C++ function), 132
 ds18b20_module_init (C++ function), 132
 ds3231 (module), 133
 ds3231_driver_t (C++ class), 133
 ds3231_driver_t::i2c_p (C++ member), 134
 ds3231_get_date (C++ function), 133
 ds3231_init (C++ function), 133
 ds3231_set_date (C++ function), 133

E

- E2BIG (C macro), 103
- EACCES (C macro), 103
- EADDRINUSE (C macro), 107
- EADDRNOTAVAIL (C macro), 107
- EADV (C macro), 105
- EAFNOSUPBOARD (C macro), 107
- EAGAIN (C macro), 103
- EALREADY (C macro), 108
- EBADE (C macro), 105
- EBADF (C macro), 103
- EBADFD (C macro), 106
- EBADMSG (C macro), 106
- EBADR (C macro), 105
- EBADRQC (C macro), 105
- EBADSLT (C macro), 105
- EBFONT (C macro), 105
- EBTASSERT (C macro), 108
- EBUSY (C macro), 103
- ECANCELED (C macro), 108
- ECHILD (C macro), 103
- ECHRNG (C macro), 104
- ECOMM (C macro), 105
- ECONNABORTED (C macro), 107
- ECONNREFUSED (C macro), 107
- ECONNRESET (C macro), 107
- EDEADLK (C macro), 104
- EDEADLOCK (C macro), 105
- EDESTADDRREQ (C macro), 106
- EDOM (C macro), 104
- EDOTDOT (C macro), 106
- EDQUOT (C macro), 108
- EEXIST (C macro), 103
- EFAULT (C macro), 103
- EFBIG (C macro), 103
- EHOSTDOWN (C macro), 107
- EHOSTUNREACH (C macro), 108
- EIDRM (C macro), 104
- EILSEQ (C macro), 106
- EINPROGRESS (C macro), 108
- EINTR (C macro), 102
- EINVAL (C macro), 103
- EIO (C macro), 102
- EISCONN (C macro), 107
- EISDIR (C macro), 103
- EISNAM (C macro), 108
- EKEYEXPIRED (C macro), 108
- EKEYREJECTED (C macro), 108
- EKEYREVOKED (C macro), 108
- EL2HLT (C macro), 105
- EL2NSYNC (C macro), 104
- EL3HLT (C macro), 104
- EL3RST (C macro), 104
- ELIBACC (C macro), 106
- ELIBBAD (C macro), 106
- ELIBEXEC (C macro), 106
- ELIBMAX (C macro), 106
- ELIBSCN (C macro), 106
- ELNRNG (C macro), 104
- ELOOP (C macro), 104
- EMEDIUMTYPE (C macro), 108
- EMFILE (C macro), 103
- EMLINK (C macro), 104
- EMSGSIZE (C macro), 106
- EMULTIHOP (C macro), 106
- ENAMETOOLONG (C macro), 104
- ENAVAIL (C macro), 108
- ENDPOINT_ATTRIBUTES_SYNCHRONISATION_TYPE (C macro), 170
- ENDPOINT_ATTRIBUTES_TRANSFER_TYPE (C macro), 170
- ENDPOINT_ATTRIBUTES_TRANSFER_TYPE_BULK (C macro), 170
- ENDPOINT_ATTRIBUTES_TRANSFER_TYPE_CONTROL (C macro), 170
- ENDPOINT_ATTRIBUTES_TRANSFER_TYPE_INTERRUPT (C macro), 171
- ENDPOINT_ATTRIBUTES_TRANSFER_TYPE_ISOCHRONOUS (C macro), 170
- ENDPOINT_ATTRIBUTES_USAGE_TYPE (C macro), 170
- ENDPOINT_ENDPOINT_ADDRESS_DIRECTION (C macro), 170
- ENDPOINT_ENDPOINT_ADDRESS_NUMBER (C macro), 170
- ENETDOWN (C macro), 107
- ENETRESET (C macro), 107
- ENETUNREACH (C macro), 107
- ENFILE (C macro), 103
- ENOANO (C macro), 105
- ENBUFS (C macro), 107
- ENOC SI (C macro), 105
- ENODATA (C macro), 105
- ENODEV (C macro), 103
- ENOENT (C macro), 102
- ENOEXEC (C macro), 103
- ENOKEY (C macro), 108
- ENOLCK (C macro), 104
- ENOLINK (C macro), 105
- ENOMEDIUM (C macro), 108
- ENOMEM (C macro), 103
- ENOMSG (C macro), 104
- ENONET (C macro), 105
- ENOPKG (C macro), 105
- ENOPROTOOPT (C macro), 107
- ENOSPC (C macro), 104
- ENOSR (C macro), 105
- ENOSTR (C macro), 105

- ENOSYS (C macro), 104
- ENOTBLK (C macro), 103
- ENOTCONN (C macro), 107
- ENOTDIR (C macro), 103
- ENOTEMPTY (C macro), 104
- ENOTNAM (C macro), 108
- ENOTSOCK (C macro), 106
- ENOTTY (C macro), 103
- ENOTUNIQ (C macro), 106
- ENXIO (C macro), 102
- EOPNOTSUPP (C macro), 107
- EOVERFLOW (C macro), 106
- EPERM (C macro), 102
- EPFNOSUPBOARD (C macro), 107
- EPIPE (C macro), 104
- EPROTO (C macro), 106
- EPROTONOSUPBOARD (C macro), 107
- EPROTOTYPE (C macro), 106
- ERANGE (C macro), 104
- EREMCHG (C macro), 106
- EREMOTE (C macro), 105
- EREMOTEIO (C macro), 108
- ERESTART (C macro), 106
- EROFS (C macro), 104
- errno (module), 102
- ESHUTDOWN (C macro), 107
- ESOCKTNOSUPBOARD (C macro), 107
- esp01 (module), 322
- esp12e (module), 323
- esp32 (module), 335
- esp8266 (module), 336
- esp_wifi (module), 134
- esp_wifi_dhcp_status_running_t (C++ class), 138
- esp_wifi_dhcp_status_stopped_t (C++ class), 138
- esp_wifi_dhcp_status_t (C++ type), 138
- esp_wifi_get_op_mode (C++ function), 138
- esp_wifi_get_phy_mode (C++ function), 139
- esp_wifi_module_init (C++ function), 138
- esp_wifi_op_mode_max_t (C++ class), 138
- esp_wifi_op_mode_null_t (C++ class), 138
- esp_wifi_op_mode_softap_t (C++ class), 138
- esp_wifi_op_mode_station_softap_t (C++ class), 138
- esp_wifi_op_mode_station_t (C++ class), 138
- esp_wifi_op_mode_t (C++ type), 138
- esp_wifi_phy_mode_11b_t (C++ class), 138
- esp_wifi_phy_mode_11g_t (C++ class), 138
- esp_wifi_phy_mode_11n_t (C++ class), 138
- esp_wifi_phy_mode_t (C++ type), 138
- esp_wifi_print (C++ function), 139
- esp_wifi_set_op_mode (C++ function), 138
- esp_wifi_set_phy_mode (C++ function), 138
- esp_wifi_softap (module), 134
- esp_wifi_softap_dhcp_server_start (C++ function), 135
- esp_wifi_softap_dhcp_server_status (C++ function), 135
- esp_wifi_softap_dhcp_server_stop (C++ function), 135
- esp_wifi_softap_get_ip_info (C++ function), 135
- esp_wifi_softap_get_number_of_connected_stations (C++ function), 135
- esp_wifi_softap_get_station_info (C++ function), 135
- esp_wifi_softap_init (C++ function), 134
- esp_wifi_softap_set_ip_info (C++ function), 134
- esp_wifi_softap_station_info_t (C++ class), 135
- esp_wifi_softap_station_info_t::bssid (C++ member), 135
- esp_wifi_softap_station_info_t::ip_address (C++ member), 135
- esp_wifi_station (module), 136
- esp_wifi_station_connect (C++ function), 136
- esp_wifi_station_dhcp_client_start (C++ function), 137
- esp_wifi_station_dhcp_client_status (C++ function), 137
- esp_wifi_station_dhcp_client_stop (C++ function), 137
- esp_wifi_station_disconnect (C++ function), 136
- esp_wifi_station_get_connect_status (C++ function), 137
- esp_wifi_station_get_ip_info (C++ function), 137
- esp_wifi_station_get_reconnect_policy (C++ function), 137
- esp_wifi_station_init (C++ function), 136
- esp_wifi_station_set_ip_info (C++ function), 136
- esp_wifi_station_set_reconnect_policy (C++ function), 137
- esp_wifi_station_status_connect_fail_t (C++ class), 136
- esp_wifi_station_status_connecting_t (C++ class), 136
- esp_wifi_station_status_got_ip_t (C++ class), 136
- esp_wifi_station_status_idle_t (C++ class), 136
- esp_wifi_station_status_no_ap_found_t (C++ class), 136
- esp_wifi_station_status_t (C++ type), 136
- esp_wifi_station_status_wrong_password_t (C++ class), 136
- ESPIPE (C macro), 104
- ESRCH (C macro), 102
- ESRMNT (C macro), 105
- ESTACK (C macro), 108
- ESTALE (C macro), 108
- ESTRPIPE (C macro), 106
- ETIME (C macro), 105
- ETIMEDOUT (C macro), 107
- ETOOMANYREFS (C macro), 107
- ETXTBSY (C macro), 103
- EUCLEAN (C macro), 108
- EUNATCH (C macro), 105
- EUSERS (C macro), 106
- event (module), 195
- event_init (C++ function), 195
- event_read (C++ function), 195
- event_size (C++ function), 196
- event_t (C++ class), 196
- event_t::base (C++ member), 196
- event_t::mask (C++ member), 196

- event_write (C++ function), 196
 - event_write_isr (C++ function), 196
 - EWOULDBLOCK (C macro), 104
 - EXDEV (C macro), 103
 - EXFULL (C macro), 105
 - exti (module), 139
 - exti_a0_dev (C macro), 314
 - exti_a10_dev (C macro), 314
 - exti_a11_dev (C macro), 314
 - exti_a1_dev (C macro), 314
 - exti_a2_dev (C macro), 314
 - exti_a3_dev (C macro), 314
 - exti_a4_dev (C macro), 314
 - exti_a5_dev (C macro), 314
 - exti_a6_dev (C macro), 314
 - exti_a7_dev (C macro), 314
 - exti_a8_dev (C macro), 314
 - exti_a9_dev (C macro), 314
 - exti_clear (C++ function), 140
 - exti_d0_dev (C macro), 312
 - exti_d10_dev (C macro), 312
 - exti_d11_dev (C macro), 313
 - exti_d12_dev (C macro), 313
 - exti_d13_dev (C macro), 313
 - exti_d14_dev (C macro), 313
 - exti_d15_dev (C macro), 313
 - exti_d16_dev (C macro), 313
 - exti_d17_dev (C macro), 313
 - exti_d18_dev (C macro), 313, 317
 - exti_d19_dev (C macro), 313, 317
 - exti_d1_dev (C macro), 312
 - exti_d20_dev (C macro), 313, 317
 - exti_d21_dev (C macro), 313, 317
 - exti_d22_dev (C macro), 313
 - exti_d23_dev (C macro), 313
 - exti_d24_dev (C macro), 313
 - exti_d25_dev (C macro), 313
 - exti_d26_dev (C macro), 313
 - exti_d27_dev (C macro), 313
 - exti_d28_dev (C macro), 313
 - exti_d29_dev (C macro), 313
 - exti_d2_dev (C macro), 312, 317–319, 321
 - exti_d30_dev (C macro), 313
 - exti_d31_dev (C macro), 313
 - exti_d32_dev (C macro), 313
 - exti_d33_dev (C macro), 313
 - exti_d34_dev (C macro), 313
 - exti_d35_dev (C macro), 313
 - exti_d36_dev (C macro), 313
 - exti_d37_dev (C macro), 313
 - exti_d38_dev (C macro), 313
 - exti_d39_dev (C macro), 313
 - exti_d3_dev (C macro), 312, 317–319, 321
 - exti_d40_dev (C macro), 313
 - exti_d41_dev (C macro), 313
 - exti_d42_dev (C macro), 313
 - exti_d43_dev (C macro), 313
 - exti_d44_dev (C macro), 313
 - exti_d45_dev (C macro), 313
 - exti_d46_dev (C macro), 313
 - exti_d47_dev (C macro), 314
 - exti_d48_dev (C macro), 314
 - exti_d49_dev (C macro), 314
 - exti_d4_dev (C macro), 312
 - exti_d50_dev (C macro), 314
 - exti_d51_dev (C macro), 314
 - exti_d52_dev (C macro), 314
 - exti_d53_dev (C macro), 314
 - exti_d5_dev (C macro), 312
 - exti_d6_dev (C macro), 312
 - exti_d7_dev (C macro), 312
 - exti_d8_dev (C macro), 312
 - exti_d9_dev (C macro), 312
 - exti_dac0_dev (C macro), 314
 - exti_dac1_dev (C macro), 314
 - exti_device (C++ member), 140
 - EXTI_DEVICE_MAX (C macro), 333–336
 - exti_init (C++ function), 139
 - exti_led_dev (C macro), 314
 - exti_module_init (C++ function), 139
 - exti_start (C++ function), 140
 - exti_stop (C++ function), 140
 - EXTI_TRIGGER_BOTH_EDGES (C macro), 139
 - EXTI_TRIGGER_FALLING_EDGE (C macro), 139
 - EXTI_TRIGGER_RISING_EDGE (C macro), 139
- ## F
- fat16 (module), 203
 - fat16_cache16_t (C++ type), 213
 - fat16_cache16_t::data (C++ member), 213
 - fat16_cache16_t::dir (C++ member), 213
 - fat16_cache16_t::fat (C++ member), 213
 - fat16_cache16_t::fbs (C++ member), 213
 - fat16_cache16_t::mbr (C++ member), 213
 - fat16_cache_t (C++ class), 213
 - fat16_cache_t::block_number (C++ member), 213
 - fat16_cache_t::buffer (C++ member), 213
 - fat16_cache_t::dirty (C++ member), 213
 - fat16_cache_t::mirror_block (C++ member), 213
 - fat16_date_t (C++ type), 209
 - fat16_date_t::as_uint16 (C++ member), 209
 - fat16_date_t::day (C++ member), 209
 - fat16_date_t::month (C++ member), 209
 - fat16_date_t::year (C++ member), 209
 - fat16_dir_close (C++ function), 208
 - fat16_dir_entry_t (C++ class), 214
 - fat16_dir_entry_t::is_dir (C++ member), 214
 - fat16_dir_entry_t::latest_mod_date (C++ member), 214

fat16_dir_entry_t::name (C++ member), 214
fat16_dir_entry_t::size (C++ member), 214
fat16_dir_open (C++ function), 208
fat16_dir_read (C++ function), 208
fat16_dir_t (C++ class), 214
fat16_dir_t::file (C++ member), 214
fat16_dir_t::root_index (C++ member), 214
FAT16_EOF (C macro), 204
fat16_file_close (C++ function), 206
fat16_file_open (C++ function), 206
fat16_file_read (C++ function), 207
fat16_file_seek (C++ function), 207
fat16_file_size (C++ function), 208
fat16_file_sync (C++ function), 208
fat16_file_t (C++ class), 214
fat16_file_t::cur_cluster (C++ member), 214
fat16_file_t::cur_position (C++ member), 214
fat16_file_t::dir_entry_block (C++ member), 214
fat16_file_t::dir_entry_index (C++ member), 214
fat16_file_t::fat16_p (C++ member), 214
fat16_file_t::file_size (C++ member), 214
fat16_file_t::first_cluster (C++ member), 214
fat16_file_t::flags (C++ member), 214
fat16_file_tell (C++ function), 207
fat16_file_truncate (C++ function), 207
fat16_file_write (C++ function), 207
fat16_format (C++ function), 206
fat16_init (C++ function), 205
fat16_mount (C++ function), 206
fat16_print (C++ function), 206
fat16_read_t (C++ type), 205
FAT16_SEEK_CUR (C macro), 204
FAT16_SEEK_END (C macro), 204
FAT16_SEEK_SET (C macro), 204
fat16_stat (C++ function), 208
fat16_stat_t (C++ class), 214
fat16_stat_t::is_dir (C++ member), 215
fat16_stat_t::size (C++ member), 215
fat16_t (C++ class), 213
fat16_t::arg_p (C++ member), 213
fat16_t::blocks_per_cluster (C++ member), 214
fat16_t::blocks_per_fat (C++ member), 214
fat16_t::cache (C++ member), 214
fat16_t::cluster_count (C++ member), 214
fat16_t::data_start_block (C++ member), 214
fat16_t::fat_count (C++ member), 214
fat16_t::fat_start_block (C++ member), 214
fat16_t::partition (C++ member), 213
fat16_t::read (C++ member), 213
fat16_t::root_dir_entry_count (C++ member), 214
fat16_t::root_dir_start_block (C++ member), 214
fat16_t::volume_start_block (C++ member), 214
fat16_t::write (C++ member), 213
fat16_time_t (C++ type), 209
fat16_time_t::as_uint16 (C++ member), 209
fat16_time_t::hours (C++ member), 209
fat16_time_t::minutes (C++ member), 209
fat16_time_t::seconds (C++ member), 209
fat16_unmount (C++ function), 206
fat16_write_t (C++ type), 205
fat_t (C++ type), 205
fbs_t (C++ class), 211
fbs_t::boot_code (C++ member), 212
fbs_t::boot_sector_sig (C++ member), 212
fbs_t::boot_signature (C++ member), 211
fbs_t::bpb (C++ member), 211
fbs_t::drive_number (C++ member), 211
fbs_t::file_system_type (C++ member), 212
fbs_t::jmp_to_boot_code (C++ member), 211
fbs_t::oem_name (C++ member), 211
fbs_t::reserved1 (C++ member), 211
fbs_t::volume_label (C++ member), 212
fbs_t::volume_serial_number (C++ member), 212
FE0 (C macro), 335
fifo (module), 283
FIFO_DEFINE_TEMPLATE (C macro), 283
fifo_get (C++ function), 284
fifo_init (C++ function), 283
fifo_put (C++ function), 283
fifo_t (C++ class), 284
fifo_t::buf_p (C++ member), 284
fifo_t::max (C++ member), 284
fifo_t::rdpos (C++ member), 284
fifo_t::wrpos (C++ member), 284
flash (module), 140
flash_0_dev (C macro), 314, 322, 323, 327, 328, 331, 333
flash_device (C++ member), 141
FLASH_DEVICE_MAX (C macro), 335–338
flash_erase (C++ function), 141
flash_init (C++ function), 141
flash_module_init (C++ function), 140
flash_read (C++ function), 141
flash_write (C++ function), 141
fs (module), 215
FS_APPEND (C macro), 216
fs_auto_complete (C++ function), 221
fs_call (C++ function), 218
fs_callback_t (C++ type), 217
fs_close (C++ function), 218
fs_command_deregister (C++ function), 223
fs_command_register (C++ function), 222
fs_command_t (C++ class), 225
fs_command_t::arg_p (C++ member), 226
fs_command_t::callback (C++ member), 226
fs_command_t::next_p (C++ member), 226
fs_counter_deregister (C++ function), 223
fs_counter_increment (C++ function), 223
fs_counter_register (C++ function), 223

fs_counter_t (C++ class), 226
 fs_counter_t::command (C++ member), 226
 fs_counter_t::next_p (C++ member), 226
 FS_CREAT (C macro), 216
 fs_dir_close (C++ function), 220
 fs_dir_entry_t (C++ class), 226
 fs_dir_entry_t::latest_mod_date (C++ member), 226
 fs_dir_entry_t::name (C++ member), 226
 fs_dir_entry_t::size (C++ member), 226
 fs_dir_entry_t::type (C++ member), 226
 fs_dir_open (C++ function), 219
 fs_dir_read (C++ function), 220
 fs_dir_t (C++ class), 226
 fs_dir_t::fat16 (C++ member), 226
 fs_dir_t::filesystem_p (C++ member), 226
 fs_dir_t::spiffs (C++ member), 226
 FS_EXCL (C macro), 216
 fs_file_t (C++ class), 225
 fs_file_t::fat16 (C++ member), 225
 fs_file_t::filesystem_p (C++ member), 225
 fs_file_t::spiffs (C++ member), 225
 fs_filesystem_deregister (C++ function), 222
 fs_filesystem_fat16_t (C++ class), 225
 fs_filesystem_fat16_t::fat16_p (C++ member), 225
 fs_filesystem_init_fat16 (C++ function), 221
 fs_filesystem_init_spiffs (C++ function), 222
 fs_filesystem_register (C++ function), 222
 fs_filesystem_spiffs_config_t (C++ class), 224
 fs_filesystem_spiffs_config_t::buf_p (C++ member), 224
 fs_filesystem_spiffs_config_t::config_p (C++ member), 224
 fs_filesystem_spiffs_config_t::size (C++ member), 224
 fs_filesystem_spiffs_config_t::workspace_p (C++ member), 224
 fs_filesystem_t (C++ class), 225
 fs_filesystem_t::fat16_p (C++ member), 225
 fs_filesystem_t::name_p (C++ member), 225
 fs_filesystem_t::next_p (C++ member), 225
 fs_filesystem_t::spiffs_p (C++ member), 225
 fs_filesystem_t::type (C++ member), 225
 fs_format (C++ function), 220
 fs_list (C++ function), 221
 fs_ls (C++ function), 220
 fs_merge (C++ function), 221
 fs_mkdir (C++ function), 220
 fs_module_init (C++ function), 217
 fs_open (C++ function), 218
 fs_parameter_deregister (C++ function), 224
 fs_parameter_int_print (C++ function), 224
 fs_parameter_int_set (C++ function), 224
 fs_parameter_print_callback_t (C++ type), 217
 fs_parameter_register (C++ function), 224
 fs_parameter_set_callback_t (C++ type), 217
 fs_parameter_t (C++ class), 226
 fs_parameter_t::command (C++ member), 226
 fs_parameter_t::next_p (C++ member), 226
 fs_parameter_t::print_cb (C++ member), 226
 fs_parameter_t::set_cb (C++ member), 226
 fs_parameter_t::value_p (C++ member), 226
 FS_RDWR (C macro), 216
 FS_READ (C macro), 216
 fs_read (C++ function), 218
 fs_read_line (C++ function), 218
 fs_remove (C++ function), 220
 fs_seek (C++ function), 219
 FS_SEEK_CUR (C macro), 216
 FS_SEEK_END (C macro), 216
 FS_SEEK_SET (C macro), 216
 fs_split (C++ function), 221
 fs_stat (C++ function), 220
 fs_stat_t (C++ class), 225
 fs_stat_t::size (C++ member), 225
 fs_stat_t::type (C++ member), 225
 FS_SYNC (C macro), 216
 fs_tell (C++ function), 219
 FS_TRUNC (C macro), 216
 FS_TYPE_DIR (C macro), 216
 fs_type_fat16_t (C++ class), 217
 FS_TYPE_FILE (C macro), 216
 FS_TYPE_HARD_LINK (C macro), 216
 FS_TYPE_SOFT_LINK (C macro), 216
 fs_type_spiffs_t (C++ class), 217
 fs_type_t (C++ type), 217
 FS_WRITE (C macro), 216
 fs_write (C++ function), 219

H

harness (module), 274
 harness_init (C++ function), 276
 harness_run (C++ function), 276
 harness_t (C++ class), 276
 harness_t::uart (C++ member), 276
 harness_testcase_cb_t (C++ type), 275
 harness_testcase_t (C++ class), 276
 harness_testcase_t::callback (C++ member), 276
 harness_testcase_t::name_p (C++ member), 276
 hash_function_t (C++ type), 284
 hash_map (module), 284
 hash_map_add (C++ function), 285
 hash_map_bucket_t (C++ class), 285
 hash_map_bucket_t::list_p (C++ member), 285
 hash_map_entry_t (C++ class), 285
 hash_map_entry_t::key (C++ member), 285
 hash_map_entry_t::next_p (C++ member), 285
 hash_map_entry_t::value_p (C++ member), 285
 hash_map_get (C++ function), 285
 hash_map_init (C++ function), 284
 hash_map_remove (C++ function), 285

`hash_map_t` (C++ class), 285
`hash_map_t::buckets_max` (C++ member), 286
`hash_map_t::buckets_p` (C++ member), 286
`hash_map_t::entries_p` (C++ member), 286
`hash_map_t::hash` (C++ member), 286
`heap` (module), 290
`heap_alloc` (C++ function), 290
`heap_dynamic_t` (C++ class), 291
`heap_dynamic_t::free_p` (C++ member), 291
`HEAP_FIXED_SIZES_MAX` (C macro), 290
`heap_fixed_t` (C++ class), 291
`heap_fixed_t::free_p` (C++ member), 291
`heap_fixed_t::size` (C++ member), 291
`heap_free` (C++ function), 290
`heap_init` (C++ function), 290
`heap_share` (C++ function), 290
`heap_t` (C++ class), 291
`heap_t::buf_p` (C++ member), 291
`heap_t::dynamic` (C++ member), 291
`heap_t::fixed` (C++ member), 291
`heap_t::next_p` (C++ member), 291
`heap_t::size` (C++ member), 291
`http_server` (module), 240
`http_server_connection_state_allocated_t` (C++ class), 241
`http_server_connection_state_free_t` (C++ class), 241
`http_server_connection_state_t` (C++ type), 241
`http_server_connection_t` (C++ class), 243
`http_server_connection_t::buf_p` (C++ member), 243
`http_server_connection_t::events` (C++ member), 243
`http_server_connection_t::id_p` (C++ member), 243
`http_server_connection_t::name_p` (C++ member), 243
`http_server_connection_t::self_p` (C++ member), 243
`http_server_connection_t::size` (C++ member), 243
`http_server_connection_t::socket` (C++ member), 243
`http_server_connection_t::state` (C++ member), 243
`http_server_content_type_t` (C++ type), 240
`http_server_content_type_text_html_t` (C++ class), 240
`http_server_content_type_text_plain_t` (C++ class), 240
`http_server_init` (C++ function), 241
`http_server_listener_t` (C++ class), 242
`http_server_listener_t::address_p` (C++ member), 243
`http_server_listener_t::buf_p` (C++ member), 243
`http_server_listener_t::id_p` (C++ member), 243
`http_server_listener_t::name_p` (C++ member), 243
`http_server_listener_t::port` (C++ member), 243
`http_server_listener_t::size` (C++ member), 243
`http_server_listener_t::socket` (C++ member), 243
`http_server_request_action_get_t` (C++ class), 240
`http_server_request_action_post_t` (C++ class), 240
`http_server_request_action_t` (C++ type), 240
`http_server_request_t` (C++ class), 242
`http_server_request_t::action` (C++ member), 242
`http_server_request_t::path` (C++ member), 242
`http_server_request_t::present` (C++ member), 242
`http_server_request_t::value` (C++ member), 242
`http_server_response_code_200_ok_t` (C++ class), 241
`http_server_response_code_401_unauthorized_t` (C++ class), 241
`http_server_response_code_404_not_found_t` (C++ class), 241
`http_server_response_code_t` (C++ type), 240
`http_server_response_t` (C++ class), 242
`http_server_response_t::buf_p` (C++ member), 242
`http_server_response_t::code` (C++ member), 242
`http_server_response_t::size` (C++ member), 242
`http_server_response_t::type` (C++ member), 242
`http_server_response_write` (C++ function), 242
`http_server_route_callback_t` (C++ type), 240
`http_server_route_t` (C++ class), 243
`http_server_route_t::callback` (C++ member), 243
`http_server_route_t::path_p` (C++ member), 243
`http_server_start` (C++ function), 241
`http_server_stop` (C++ function), 241
`http_server_t` (C++ class), 243
`http_server_t::connections_p` (C++ member), 244
`http_server_t::events` (C++ member), 244
`http_server_t::listener_p` (C++ member), 243
`http_server_t::on_no_route` (C++ member), 243
`http_server_t::root_path_p` (C++ member), 243
`http_server_t::routes_p` (C++ member), 243
`http_websocket_client` (module), 244
`http_websocket_client_connect` (C++ function), 244
`http_websocket_client_disconnect` (C++ function), 244
`http_websocket_client_init` (C++ function), 244
`http_websocket_client_read` (C++ function), 244
`http_websocket_client_t` (C++ class), 245
`http_websocket_client_t::host_p` (C++ member), 245
`http_websocket_client_t::left` (C++ member), 245
`http_websocket_client_t::path_p` (C++ member), 245
`http_websocket_client_t::port` (C++ member), 245
`http_websocket_client_t::socket` (C++ member), 245
`http_websocket_client_write` (C++ function), 245
`http_websocket_server` (module), 245
`http_websocket_server_handshake` (C++ function), 246
`http_websocket_server_init` (C++ function), 245
`http_websocket_server_read` (C++ function), 246
`http_websocket_server_t` (C++ class), 246
`http_websocket_server_t::socket_p` (C++ member), 246
`http_websocket_server_write` (C++ function), 246

I

`i2c` (module), 142
`i2c_0_dev` (C macro), 317, 318, 320, 321, 331, 333
`i2c_1_dev` (C macro), 331, 333
`I2C_BAUDRATE_100KBPS` (C macro), 142
`I2C_BAUDRATE_1MBPS` (C macro), 142
`I2C_BAUDRATE_3_2MBPS` (C macro), 142

I2C_BAUDRATE_400KBPS (C macro), 142
 i2c_dev (C macro), 326
 i2c_device (C++ member), 144
 I2C_DEVICE_MAX (C macro), 334, 335, 337, 338
 i2c_init (C++ function), 142
 i2c_module_init (C++ function), 142
 i2c_read (C++ function), 143
 i2c_scan (C++ function), 143
 i2c_slave_read (C++ function), 144
 i2c_slave_start (C++ function), 143
 i2c_slave_stop (C++ function), 144
 i2c_slave_write (C++ function), 144
 i2c_soft (module), 144
 i2c_soft_driver_t (C++ class), 146
 i2c_soft_driver_t::baudrate (C++ member), 146
 i2c_soft_driver_t::baudrate_us (C++ member), 146
 i2c_soft_driver_t::clock_stretching_sleep_us (C++ member), 146
 i2c_soft_driver_t::max_clock_stretching_us (C++ member), 146
 i2c_soft_driver_t::scl_p (C++ member), 146
 i2c_soft_driver_t::sda_p (C++ member), 146
 i2c_soft_init (C++ function), 144
 i2c_soft_module_init (C++ function), 144
 i2c_soft_read (C++ function), 145
 i2c_soft_scan (C++ function), 146
 i2c_soft_start (C++ function), 145
 i2c_soft_stop (C++ function), 145
 i2c_soft_write (C++ function), 145
 i2c_start (C++ function), 142
 i2c_stop (C++ function), 143
 i2c_write (C++ function), 143
 inet (module), 246
 inet_addr_t (C++ class), 248
 inet_addr_t::ip (C++ member), 248
 inet_addr_t::port (C++ member), 248
 inet_aton (C++ function), 247
 inet_checksum (C++ function), 247
 inet_if_ip_info_t (C++ class), 248
 inet_if_ip_info_t::address (C++ member), 248
 inet_if_ip_info_t::gateway (C++ member), 248
 inet_if_ip_info_t::netmask (C++ member), 248
 inet_ip_addr_t (C++ class), 247
 inet_ip_addr_t::number (C++ member), 248
 inet_module_init (C++ function), 247
 inet_ntoa (C++ function), 247

J

json (module), 300
 JSON_ARRAY (C++ class), 301
 json_array_get (C++ function), 303
 json_dump (C++ function), 302
 json_dumps (C++ function), 302
 json_err_t (C++ type), 301

JSON_ERROR_INVALID (C++ class), 301
 JSON_ERROR_NOMEM (C++ class), 301
 JSON_ERROR_PART (C++ class), 301
 json_init (C++ function), 301
 JSON_OBJECT (C++ class), 301
 json_object_get (C++ function), 302
 json_object_get_primitive (C++ function), 302
 json_parse (C++ function), 301
 JSON_PRIMITIVE (C++ class), 301
 json_root (C++ function), 302
 JSON_STRING (C++ class), 301
 json_t (C++ class), 304
 json_t::num_tokens (C++ member), 304
 json_t::pos (C++ member), 304
 json_t::tokens_p (C++ member), 304
 json_t::toknext (C++ member), 304
 json_t::toksuper (C++ member), 304
 json_tok_t (C++ class), 304
 json_tok_t::buf_p (C++ member), 304
 json_tok_t::num_tokens (C++ member), 304
 json_tok_t::size (C++ member), 304
 json_tok_t::type (C++ member), 304
 json_token_array (C++ function), 303
 json_token_false (C++ function), 303
 json_token_null (C++ function), 303
 json_token_number (C++ function), 303
 json_token_object (C++ function), 303
 json_token_string (C++ function), 304
 json_token_true (C++ function), 303
 json_type_t (C++ type), 301
 JSON_UNDEFINED (C++ class), 301

L

linux (module), 324, 336
 list (module), 286
 list_next_t (C++ class), 287
 list_next_t::next_p (C++ member), 287
 list_singly_linked_t (C++ class), 287
 list_singly_linked_t::head_p (C++ member), 287
 list_singly_linked_t::tail_p (C++ member), 287
 LIST_SL_ADD_HEAD (C macro), 286
 LIST_SL_ADD_TAIL (C macro), 286
 LIST_SL_INIT (C macro), 286
 LIST_SL_INIT_STRUCT (C macro), 286
 LIST_SL_ITERATOR_INIT (C macro), 287
 LIST_SL_ITERATOR_NEXT (C macro), 287
 list_sl_iterator_t (C++ class), 287
 list_sl_iterator_t::next_p (C++ member), 287
 LIST_SL_PEEK_HEAD (C macro), 286
 LIST_SL_REMOVE_ELEM (C macro), 287
 LIST_SL_REMOVE_HEAD (C macro), 286
 log (module), 276
 log_add_handler (C++ function), 279
 log_add_object (C++ function), 280

LOG_ALL (C macro), 278
 LOG_DEBUG (C macro), 278
 LOG_ERROR (C macro), 278
 LOG_FATAL (C macro), 278
 log_handler_init (C++ function), 279
 log_handler_t (C++ class), 280
 log_handler_t::chout_p (C++ member), 280
 log_handler_t::next_p (C++ member), 280
 LOG_INFO (C macro), 278
 LOG_MASK (C macro), 278
 log_module_init (C++ function), 278
 LOG_NONE (C macro), 278
 log_object_get_log_mask (C++ function), 278
 log_object_init (C++ function), 278
 log_object_is_enabled_for (C++ function), 279
 log_object_print (C++ function), 279
 log_object_set_log_mask (C++ function), 278
 log_object_t (C++ class), 280
 log_object_t::mask (C++ member), 280
 log_object_t::name_p (C++ member), 280
 log_object_t::next_p (C++ member), 280
 log_remove_handler (C++ function), 279
 log_remove_object (C++ function), 280
 log_set_default_handler_output_channel (C++ function), 280
 LOG_UPTO (C macro), 278
 LOG_WARNING (C macro), 278

M

MAX (C macro), 122
 mbr_t (C++ class), 212
 mbr_t::codeArea (C++ member), 212
 mbr_t::diskSignature (C++ member), 212
 mbr_t::mbr_sig (C++ member), 212
 mbr_t::part (C++ member), 212
 mbr_t::usuallyZero (C++ member), 212
 mcp2515 (module), 146
 mcp2515_driver_t (C++ class), 148
 mcp2515_driver_t::chin_p (C++ member), 148
 mcp2515_driver_t::chout (C++ member), 148
 mcp2515_driver_t::exti (C++ member), 148
 mcp2515_driver_t::isr_sem (C++ member), 148
 mcp2515_driver_t::mode (C++ member), 148
 mcp2515_driver_t::speed (C++ member), 148
 mcp2515_driver_t::spi (C++ member), 148
 mcp2515_driver_t::tx_sem (C++ member), 148
 mcp2515_frame_t (C++ class), 148
 mcp2515_frame_t::data (C++ member), 148
 mcp2515_frame_t::id (C++ member), 148
 mcp2515_frame_t::rtr (C++ member), 148
 mcp2515_frame_t::size (C++ member), 148
 mcp2515_frame_t::timestamp (C++ member), 148
 mcp2515_init (C++ function), 147
 MCP2515_MODE_LOOPBACK (C macro), 146

MCP2515_MODE_NORMAL (C macro), 146
 mcp2515_read (C++ function), 147
 MCP2515_SPEED_1000KBPS (C macro), 146
 MCP2515_SPEED_500KBPS (C macro), 146
 mcp2515_start (C++ function), 147
 mcp2515_stop (C++ function), 147
 mcp2515_write (C++ function), 147
 membersof (C macro), 122
 midi (module), 307
 MIDI_BAUDRATE (C macro), 307
 MIDI_CHANNEL_PRESSURE (C macro), 307
 MIDI_CONTROL_CHANGE (C macro), 307
 MIDI_NOTE_A0 (C macro), 307
 MIDI_NOTE_A1 (C macro), 307
 MIDI_NOTE_A2 (C macro), 307
 MIDI_NOTE_A3 (C macro), 308
 MIDI_NOTE_A4 (C macro), 308
 MIDI_NOTE_A5 (C macro), 308
 MIDI_NOTE_A6 (C macro), 308
 MIDI_NOTE_A7 (C macro), 308
 MIDI_NOTE_B0 (C macro), 307
 MIDI_NOTE_B1 (C macro), 307
 MIDI_NOTE_B2 (C macro), 307
 MIDI_NOTE_B3 (C macro), 308
 MIDI_NOTE_B4 (C macro), 308
 MIDI_NOTE_B5 (C macro), 308
 MIDI_NOTE_B6 (C macro), 308
 MIDI_NOTE_B7 (C macro), 308
 MIDI_NOTE_C1 (C macro), 307
 MIDI_NOTE_C2 (C macro), 307
 MIDI_NOTE_C3 (C macro), 307
 MIDI_NOTE_C4 (C macro), 308
 MIDI_NOTE_C5 (C macro), 308
 MIDI_NOTE_C6 (C macro), 308
 MIDI_NOTE_C7 (C macro), 308
 MIDI_NOTE_C8 (C macro), 308
 MIDI_NOTE_D1 (C macro), 307
 MIDI_NOTE_D2 (C macro), 307
 MIDI_NOTE_D3 (C macro), 307
 MIDI_NOTE_D4 (C macro), 308
 MIDI_NOTE_D5 (C macro), 308
 MIDI_NOTE_D6 (C macro), 308
 MIDI_NOTE_D7 (C macro), 308
 MIDI_NOTE_E1 (C macro), 307
 MIDI_NOTE_E2 (C macro), 307
 MIDI_NOTE_E3 (C macro), 308
 MIDI_NOTE_E4 (C macro), 308
 MIDI_NOTE_E5 (C macro), 308
 MIDI_NOTE_E6 (C macro), 308
 MIDI_NOTE_E7 (C macro), 308
 MIDI_NOTE_F1 (C macro), 307
 MIDI_NOTE_F2 (C macro), 307
 MIDI_NOTE_F3 (C macro), 308
 MIDI_NOTE_F4 (C macro), 308

- MIDI_NOTE_F5 (C macro), 308
 - MIDI_NOTE_F6 (C macro), 308
 - MIDI_NOTE_F7 (C macro), 308
 - MIDI_NOTE_G1 (C macro), 307
 - MIDI_NOTE_G2 (C macro), 307
 - MIDI_NOTE_G3 (C macro), 308
 - MIDI_NOTE_G4 (C macro), 308
 - MIDI_NOTE_G5 (C macro), 308
 - MIDI_NOTE_G6 (C macro), 308
 - MIDI_NOTE_G7 (C macro), 308
 - MIDI_NOTE_MAX (C macro), 307
 - MIDI_NOTE_OFF (C macro), 307
 - MIDI_NOTE_ON (C macro), 307
 - midi_note_to_frequency (C++ function), 310
 - MIDI_PERC (C macro), 307
 - MIDI_PERC_ACOUSTIC_BASS_DRUM (C macro), 308
 - MIDI_PERC_ACOUSTIC_SNARE (C macro), 309
 - MIDI_PERC_BASS_DRUM_1 (C macro), 308
 - MIDI_PERC_CABASA (C macro), 309
 - MIDI_PERC_CHINESE_CYMBAL (C macro), 309
 - MIDI_PERC_CLAVES (C macro), 310
 - MIDI_PERC_CLOSED_HI_HAT (C macro), 309
 - MIDI_PERC_COWBELL (C macro), 309
 - MIDI_PERC_CRASH_CYMBAL_1 (C macro), 309
 - MIDI_PERC_CRASH_CYMBAL_2 (C macro), 309
 - MIDI_PERC_ELECTRIC_SNARE (C macro), 309
 - MIDI_PERC_HAND_CLAP (C macro), 309
 - MIDI_PERC_HI_BONGO (C macro), 309
 - MIDI_PERC_HI_MID_TOM (C macro), 309
 - MIDI_PERC_HI_WOOD_BLOCK (C macro), 310
 - MIDI_PERC_HIGH_AGOGO (C macro), 309
 - MIDI_PERC_HIGH_FLOOR_TOM (C macro), 309
 - MIDI_PERC_HIGH_TIMBALE (C macro), 309
 - MIDI_PERC_HIGH_TOM (C macro), 309
 - MIDI_PERC_LONG_GUIRO (C macro), 310
 - MIDI_PERC_LONG_WHISTLE (C macro), 309
 - MIDI_PERC_LOW_AGOGO (C macro), 309
 - MIDI_PERC_LOW_BONGO (C macro), 309
 - MIDI_PERC_LOW_CONGA (C macro), 309
 - MIDI_PERC_LOW_FLOOR_TOM (C macro), 309
 - MIDI_PERC_LOW_MID_TOM (C macro), 309
 - MIDI_PERC_LOW_TIMBALE (C macro), 309
 - MIDI_PERC_LOW_TOM (C macro), 309
 - MIDI_PERC_LOW_WOOD_BLOCK (C macro), 310
 - MIDI_PERC_MARACAS (C macro), 309
 - MIDI_PERC_MUTE_CUICA (C macro), 310
 - MIDI_PERC_MUTE_HI_CONGA (C macro), 309
 - MIDI_PERC_MUTE_TRIANGLE (C macro), 310
 - MIDI_PERC_OPEN_CUICA (C macro), 310
 - MIDI_PERC_OPEN_HI_CONGA (C macro), 309
 - MIDI_PERC_OPEN_HI_HAT (C macro), 309
 - MIDI_PERC_OPEN_TRIANGLE (C macro), 310
 - MIDI_PERC_PEDAL_HI_HAT (C macro), 309
 - MIDI_PERC_RIDE_BELL (C macro), 309
 - MIDI_PERC_RIDE_CYMBAL_1 (C macro), 309
 - MIDI_PERC_RIDE_CYMBAL_2 (C macro), 309
 - MIDI_PERC_SHORT_GUIRO (C macro), 310
 - MIDI_PERC_SHORT_WHISTLE (C macro), 309
 - MIDI_PERC_SIDE_STICK (C macro), 309
 - MIDI_PERC_SPLASH_CYMBAL (C macro), 309
 - MIDI_PERC_TAMBOURINE (C macro), 309
 - MIDI_PERC_VIBRASLAP (C macro), 309
 - MIDI_PITCH_BEND_CHANGE (C macro), 307
 - MIDI_POLYPHONIC_KEY_PRESSURE (C macro), 307
 - MIDI_PROGRAM_CHANGE (C macro), 307
 - MIDI_SET_INTRUMENT (C macro), 307
 - MIN (C macro), 122
 - mqtt_application_message_t (C++ class), 251
 - mqtt_application_message_t::buf_p (C++ member), 251
 - mqtt_application_message_t::qos (C++ member), 251
 - mqtt_application_message_t::size (C++ member), 251
 - mqtt_client (module), 248
 - mqtt_client_connect (C++ function), 250
 - mqtt_client_disconnect (C++ function), 250
 - mqtt_client_init (C++ function), 249
 - mqtt_client_main (C++ function), 249
 - mqtt_client_ping (C++ function), 250
 - mqtt_client_publish (C++ function), 250
 - mqtt_client_state_connected_t (C++ class), 249
 - mqtt_client_state_connecting_t (C++ class), 249
 - mqtt_client_state_disconnected_t (C++ class), 249
 - mqtt_client_state_t (C++ type), 249
 - mqtt_client_subscribe (C++ function), 250
 - mqtt_client_t (C++ class), 251
 - mqtt_client_t::data_p (C++ member), 251
 - mqtt_client_t::in (C++ member), 251
 - mqtt_client_t::in_p (C++ member), 251
 - mqtt_client_t::log_object_p (C++ member), 251
 - mqtt_client_t::name_p (C++ member), 251
 - mqtt_client_t::on_error (C++ member), 251
 - mqtt_client_t::on_publish (C++ member), 251
 - mqtt_client_t::out (C++ member), 251
 - mqtt_client_t::out_p (C++ member), 251
 - mqtt_client_t::state (C++ member), 251
 - mqtt_client_t::type (C++ member), 251
 - mqtt_client_unsubscribe (C++ function), 250
 - mqtt_on_error_t (C++ type), 248
 - mqtt_on_publish_t (C++ type), 248
 - mqtt_qos_0_t (C++ class), 249
 - mqtt_qos_1_t (C++ class), 249
 - mqtt_qos_2_t (C++ class), 249
 - mqtt_qos_t (C++ type), 249
- N**
- nano32 (module), 325
 - network_interface (module), 251

- network_interface_add (C++ function), 256
 - network_interface_driver_esp (module), 253
 - network_interface_get_by_name (C++ function), 257
 - network_interface_get_ip_info (C++ function), 257
 - network_interface_get_ip_info_t (C++ type), 256
 - network_interface_is_up (C++ function), 257
 - network_interface_is_up_t (C++ type), 256
 - network_interface_module_init (C++ function), 256
 - network_interface_set_ip_info (C++ function), 257
 - network_interface_set_ip_info_t (C++ type), 256
 - network_interface_slip (module), 252
 - NETWORK_INTERFACE_SLIP_FRAME_SIZE_MAX (C macro), 252
 - network_interface_slip_init (C++ function), 252
 - network_interface_slip_input (C++ function), 252
 - network_interface_slip_module_init (C++ function), 252
 - NETWORK_INTERFACE_SLIP_STATE_ESCAPE (C++ class), 252
 - NETWORK_INTERFACE_SLIP_STATE_NORMAL (C++ class), 252
 - network_interface_slip_state_t (C++ type), 252
 - network_interface_slip_t (C++ class), 253
 - network_interface_slip_t::buf_p (C++ member), 253
 - network_interface_slip_t::chout_p (C++ member), 253
 - network_interface_slip_t::network_interface (C++ member), 253
 - network_interface_slip_t::pbuf_p (C++ member), 253
 - network_interface_slip_t::size (C++ member), 253
 - network_interface_slip_t::state (C++ member), 253
 - network_interface_start (C++ function), 256
 - network_interface_start_t (C++ type), 256
 - network_interface_stop (C++ function), 257
 - network_interface_stop_t (C++ type), 256
 - network_interface_t (C++ class), 257
 - network_interface_t::get_ip_info (C++ member), 258
 - network_interface_t::info (C++ member), 258
 - network_interface_t::is_up (C++ member), 258
 - network_interface_t::name_p (C++ member), 258
 - network_interface_t::netif_p (C++ member), 258
 - network_interface_t::next_p (C++ member), 258
 - network_interface_t::set_ip_info (C++ member), 258
 - network_interface_t::start (C++ member), 258
 - network_interface_t::stop (C++ member), 258
 - network_interface_wifi (module), 253
 - network_interface_wifi_driver_esp_softap (C++ member), 253
 - network_interface_wifi_driver_esp_station (C++ member), 253
 - network_interface_wifi_driver_t (C++ class), 255
 - network_interface_wifi_driver_t::get_ip_info (C++ member), 255
 - network_interface_wifi_driver_t::init (C++ member), 255
 - network_interface_wifi_driver_t::is_up (C++ member), 255
 - network_interface_wifi_driver_t::set_ip_info (C++ member), 255
 - network_interface_wifi_driver_t::start (C++ member), 255
 - network_interface_wifi_driver_t::stop (C++ member), 255
 - network_interface_wifi_get_ip_info (C++ function), 255
 - network_interface_wifi_init (C++ function), 254
 - network_interface_wifi_is_up (C++ function), 254
 - network_interface_wifi_module_init (C++ function), 254
 - network_interface_wifi_set_ip_info (C++ function), 254
 - network_interface_wifi_start (C++ function), 254
 - network_interface_wifi_stop (C++ function), 254
 - network_interface_wifi_t (C++ class), 255
 - network_interface_wifi_t::arg_p (C++ member), 255
 - network_interface_wifi_t::driver_p (C++ member), 255
 - network_interface_wifi_t::info_p (C++ member), 255
 - network_interface_wifi_t::network_interface (C++ member), 255
 - network_interface_wifi_t::password_p (C++ member), 255
 - network_interface_wifi_t::ssid_p (C++ member), 255
 - nrf24l01 (module), 148
 - nrf24l01_driver_t (C++ class), 149
 - nrf24l01_driver_t::address (C++ member), 150
 - nrf24l01_driver_t::ce (C++ member), 150
 - nrf24l01_driver_t::chin (C++ member), 150
 - nrf24l01_driver_t::chinbuf (C++ member), 150
 - nrf24l01_driver_t::exti (C++ member), 150
 - nrf24l01_driver_t::irqbuf (C++ member), 150
 - nrf24l01_driver_t::irqchan (C++ member), 150
 - nrf24l01_driver_t::spi (C++ member), 150
 - nrf24l01_driver_t::stack (C++ member), 150
 - nrf24l01_driver_t::thrd_p (C++ member), 150
 - nrf24l01_init (C++ function), 148
 - nrf24l01_module_init (C++ function), 148
 - nrf24l01_read (C++ function), 149
 - nrf24l01_start (C++ function), 149
 - nrf24l01_stop (C++ function), 149
 - nrf24l01_write (C++ function), 149
- ## O
- O_APPEND (C macro), 205
 - O_CREAT (C macro), 205
 - O_EXCL (C macro), 205
 - O_RDONLY (C macro), 204
 - O_RDWR (C macro), 204
 - O_READ (C macro), 204
 - O_SYNC (C macro), 205
 - O_TRUNC (C macro), 205
 - O_WRITE (C macro), 204
 - O_WRONLY (C macro), 204
 - owi (module), 150
 - OWI_ALARM_SEARCH (C macro), 150

owi_device_t (C++ class), 151
 owi_device_t::id (C++ member), 151
 owi_driver_t (C++ class), 151
 owi_driver_t::devices_p (C++ member), 152
 owi_driver_t::len (C++ member), 152
 owi_driver_t::nmemb (C++ member), 152
 owi_driver_t::pin (C++ member), 152
 owi_init (C++ function), 150
 OWI_MATCH_ROM (C macro), 150
 owi_read (C++ function), 151
 OWI_READ_ROM (C macro), 150
 owi_reset (C++ function), 151
 owi_search (C++ function), 151
 OWI_SEARCH_ROM (C macro), 150
 OWI_SKIP_ROM (C macro), 150
 owi_write (C++ function), 151

P

PACKED (C++ member), 157, 209
 part_t (C++ class), 209
 part_t::begin_cylinder_high (C++ member), 210
 part_t::begin_cylinder_low (C++ member), 210
 part_t::begin_head (C++ member), 210
 part_t::begin_sector (C++ member), 210
 part_t::boot (C++ member), 210
 part_t::end_cylinder_high (C++ member), 210
 part_t::end_cylinder_low (C++ member), 210
 part_t::end_head (C++ member), 210
 part_t::end_sector (C++ member), 210
 part_t::first_sector (C++ member), 210
 part_t::total_sectors (C++ member), 210
 part_t::type (C++ member), 210
 photon (module), 327
 pin (module), 152
 pin_a0_dev (C macro), 312, 316, 318–321, 323, 324, 327
 pin_a10_dev (C macro), 312, 317
 pin_a11_dev (C macro), 312, 317
 pin_a12_dev (C macro), 317
 pin_a13_dev (C macro), 317
 pin_a14_dev (C macro), 317
 pin_a15_dev (C macro), 317
 pin_a1_dev (C macro), 312, 316, 318–320, 322, 324, 327
 pin_a2_dev (C macro), 312, 316, 318–320, 322, 324, 327
 pin_a3_dev (C macro), 312, 316, 318–320, 322, 324, 327
 pin_a4_dev (C macro), 312, 316, 318, 320, 322, 324, 327
 pin_a5_dev (C macro), 312, 317, 318, 320, 322, 324, 327
 pin_a6_dev (C macro), 312, 317, 322, 324
 pin_a7_dev (C macro), 312, 317, 322, 324
 pin_a8_dev (C macro), 312, 317
 pin_a9_dev (C macro), 312, 317
 pin_adc0_dev (C macro), 326
 pin_adc10_dev (C macro), 326
 pin_adc11_dev (C macro), 326
 pin_adc12_dev (C macro), 326
 pin_adc13_dev (C macro), 326
 pin_adc14_dev (C macro), 326
 pin_adc15_dev (C macro), 326
 pin_adc16_dev (C macro), 326
 pin_adc17_dev (C macro), 326
 pin_adc18_dev (C macro), 326
 pin_adc19_dev (C macro), 326
 pin_adc3_dev (C macro), 326
 pin_adc4_dev (C macro), 326
 pin_adc5_dev (C macro), 326
 pin_adc6_dev (C macro), 326
 pin_adc7_dev (C macro), 326
 pin_d0_dev (C macro), 310, 315, 322, 323, 327
 pin_d10_dev (C macro), 311, 315, 318–321, 324
 pin_d11_dev (C macro), 311, 315, 318, 320, 321, 324
 pin_d12_dev (C macro), 311, 315, 318, 320, 321, 323, 324
 pin_d13_dev (C macro), 311, 315, 318, 320, 321, 323, 324
 pin_d14_dev (C macro), 311, 315, 319, 323
 pin_d15_dev (C macro), 311, 315, 319, 323
 pin_d16_dev (C macro), 311, 315, 319
 pin_d17_dev (C macro), 311, 315
 pin_d18_dev (C macro), 311, 315
 pin_d19_dev (C macro), 311, 315
 pin_d1_dev (C macro), 310, 315, 322, 327
 pin_d20_dev (C macro), 311, 315
 pin_d21_dev (C macro), 311, 315
 pin_d22_dev (C macro), 311, 315
 pin_d23_dev (C macro), 311, 316
 pin_d24_dev (C macro), 311, 316
 pin_d25_dev (C macro), 311, 316
 pin_d26_dev (C macro), 311, 316
 pin_d27_dev (C macro), 311, 316
 pin_d28_dev (C macro), 311, 316
 pin_d29_dev (C macro), 311, 316
 pin_d2_dev (C macro), 310, 315, 318–324, 327
 pin_d30_dev (C macro), 311, 316
 pin_d31_dev (C macro), 311, 316
 pin_d32_dev (C macro), 311, 316
 pin_d33_dev (C macro), 311, 316
 pin_d34_dev (C macro), 311, 316
 pin_d35_dev (C macro), 311, 316
 pin_d36_dev (C macro), 311, 316
 pin_d37_dev (C macro), 311, 316
 pin_d38_dev (C macro), 311, 316
 pin_d39_dev (C macro), 311, 316
 pin_d3_dev (C macro), 310, 315, 318–321, 324, 327
 pin_d40_dev (C macro), 311, 316
 pin_d41_dev (C macro), 311, 316
 pin_d42_dev (C macro), 311, 316
 pin_d43_dev (C macro), 311, 316
 pin_d44_dev (C macro), 312, 316
 pin_d45_dev (C macro), 312, 316

pin_d46_dev (C macro), 312, 316
pin_d47_dev (C macro), 312, 316
pin_d48_dev (C macro), 312, 316
pin_d49_dev (C macro), 312, 316
pin_d4_dev (C macro), 310, 315, 318–321, 323, 324, 327
pin_d50_dev (C macro), 312, 316
pin_d51_dev (C macro), 312, 316
pin_d52_dev (C macro), 312, 316
pin_d53_dev (C macro), 312, 316
pin_d5_dev (C macro), 310, 315, 318–321, 323, 324, 327
pin_d6_dev (C macro), 310, 315, 318–321, 324, 327
pin_d7_dev (C macro), 310, 315, 318–321, 324, 327
pin_d8_dev (C macro), 311, 315, 318–321, 324
pin_d9_dev (C macro), 311, 315, 318–321, 324
pin_dac0_dev (C macro), 312, 325, 327
pin_dac1_dev (C macro), 312, 325, 326, 328
pin_dac2_dev (C macro), 326
pin_device (C++ member), 154
PIN_DEVICE_BASE (C macro), 321, 324
PIN_DEVICE_MAX (C macro), 333–338
pin_device_read (C++ function), 154
pin_device_set_mode (C++ function), 153
pin_device_write_high (C++ function), 154
pin_device_write_low (C++ function), 154
pin_gpio00_dev (C macro), 325
pin_gpio01_dev (C macro), 325
pin_gpio02_dev (C macro), 325
pin_gpio03_dev (C macro), 325
pin_gpio04_dev (C macro), 325
pin_gpio05_dev (C macro), 325
pin_gpio06_dev (C macro), 325
pin_gpio07_dev (C macro), 325
pin_gpio08_dev (C macro), 325
pin_gpio09_dev (C macro), 325
pin_gpio0_dev (C macro), 322, 323
pin_gpio10_dev (C macro), 325
pin_gpio11_dev (C macro), 325
pin_gpio12_dev (C macro), 323, 325
pin_gpio13_dev (C macro), 323, 325
pin_gpio14_dev (C macro), 323, 325
pin_gpio15_dev (C macro), 323, 325
pin_gpio16_dev (C macro), 326
pin_gpio17_dev (C macro), 326
pin_gpio18_dev (C macro), 326
pin_gpio19_dev (C macro), 326
pin_gpio1_dev (C macro), 322
pin_gpio21_dev (C macro), 326
pin_gpio22_dev (C macro), 326
pin_gpio23_dev (C macro), 326
pin_gpio25_dev (C macro), 326
pin_gpio26_dev (C macro), 326
pin_gpio27_dev (C macro), 326
pin_gpio2_dev (C macro), 322, 323
pin_gpio32_dev (C macro), 326
pin_gpio33_dev (C macro), 326
pin_gpio34_dev (C macro), 326
pin_gpio35_dev (C macro), 326
pin_gpio36_dev (C macro), 326
pin_gpio39_dev (C macro), 326
pin_gpio4_dev (C macro), 323
pin_gpio5_dev (C macro), 323
pin_init (C++ function), 153
PIN_INPUT (C macro), 152
pin_ld3_dev (C macro), 333
pin_ld4_dev (C macro), 333
pin_led_dev (C macro), 312, 317–320, 322–324, 326, 327, 333
pin_module_init (C++ function), 152
PIN_OUTPUT (C macro), 152
pin_pa0_dev (C macro), 328, 331
pin_pa10_dev (C macro), 328, 331
pin_pa11_dev (C macro), 328, 331
pin_pa12_dev (C macro), 329, 331
pin_pa13_dev (C macro), 329, 331
pin_pa14_dev (C macro), 329, 331
pin_pa15_dev (C macro), 329, 332
pin_pa1_dev (C macro), 328, 331
pin_pa2_dev (C macro), 328, 331
pin_pa3_dev (C macro), 328, 331
pin_pa4_dev (C macro), 328, 331
pin_pa5_dev (C macro), 328, 331
pin_pa6_dev (C macro), 328, 331
pin_pa7_dev (C macro), 328, 331
pin_pa8_dev (C macro), 328, 331
pin_pa9_dev (C macro), 328, 331
pin_pb0_dev (C macro), 329, 332
pin_pb10_dev (C macro), 329, 332
pin_pb11_dev (C macro), 329, 332
pin_pb12_dev (C macro), 329, 332
pin_pb13_dev (C macro), 329, 332
pin_pb14_dev (C macro), 329, 332
pin_pb15_dev (C macro), 329, 332
pin_pb1_dev (C macro), 329, 332
pin_pb2_dev (C macro), 329, 332
pin_pb3_dev (C macro), 329, 332
pin_pb4_dev (C macro), 329, 332
pin_pb5_dev (C macro), 329, 332
pin_pb6_dev (C macro), 329, 332
pin_pb7_dev (C macro), 329, 332
pin_pb8_dev (C macro), 329, 332
pin_pb9_dev (C macro), 329, 332
pin_pc0_dev (C macro), 329, 332
pin_pc10_dev (C macro), 329, 332
pin_pc11_dev (C macro), 329, 332
pin_pc12_dev (C macro), 329, 332
pin_pc13_dev (C macro), 329, 332
pin_pc14_dev (C macro), 329, 332
pin_pc15_dev (C macro), 329, 332

pin_pc1_dev (C macro), 329, 332
 pin_pc2_dev (C macro), 329, 332
 pin_pc3_dev (C macro), 329, 332
 pin_pc4_dev (C macro), 329, 332
 pin_pc5_dev (C macro), 329, 332
 pin_pc6_dev (C macro), 329, 332
 pin_pc7_dev (C macro), 329, 332
 pin_pc8_dev (C macro), 329, 332
 pin_pc9_dev (C macro), 329, 332
 pin_pd0_dev (C macro), 330, 332
 pin_pd10_dev (C macro), 330
 pin_pd11_dev (C macro), 330
 pin_pd12_dev (C macro), 330
 pin_pd13_dev (C macro), 330
 pin_pd14_dev (C macro), 330
 pin_pd15_dev (C macro), 330
 pin_pd1_dev (C macro), 330, 332
 pin_pd2_dev (C macro), 330, 332
 pin_pd3_dev (C macro), 330
 pin_pd4_dev (C macro), 330
 pin_pd5_dev (C macro), 330
 pin_pd6_dev (C macro), 330
 pin_pd7_dev (C macro), 330
 pin_pd8_dev (C macro), 330
 pin_pd9_dev (C macro), 330
 pin_pe0_dev (C macro), 330
 pin_pe10_dev (C macro), 330
 pin_pe11_dev (C macro), 330
 pin_pe12_dev (C macro), 330
 pin_pe13_dev (C macro), 330
 pin_pe14_dev (C macro), 330
 pin_pe15_dev (C macro), 330
 pin_pe1_dev (C macro), 330
 pin_pe2_dev (C macro), 330
 pin_pe3_dev (C macro), 330
 pin_pe4_dev (C macro), 330
 pin_pe5_dev (C macro), 330
 pin_pe6_dev (C macro), 330
 pin_pe7_dev (C macro), 330
 pin_pe8_dev (C macro), 330
 pin_pe9_dev (C macro), 330
 pin_read (C++ function), 153
 pin_set_mode (C++ function), 153
 pin_toggle (C++ function), 153
 pin_write (C++ function), 153
 ping (module), 258
 ping_host_by_ip_address (C++ function), 258
 ping_module_init (C++ function), 258
 PRINT_FILE_LINE (C macro), 122
 pwm (module), 154
 pwm_a4_dev (C macro), 328
 pwm_a5_dev (C macro), 328
 pwm_d0_dev (C macro), 328
 pwm_d10_dev (C macro), 314, 317–319, 321, 322, 325

pwm_d11_dev (C macro), 314, 317–319, 321, 322, 325
 pwm_d12_dev (C macro), 314, 317
 pwm_d1_dev (C macro), 328
 pwm_d2_dev (C macro), 314, 317, 328
 pwm_d3_dev (C macro), 314, 317–319, 321, 322, 324, 328
 pwm_d5_dev (C macro), 314, 317
 pwm_d6_dev (C macro), 314, 317
 pwm_d7_dev (C macro), 314, 317
 pwm_d8_dev (C macro), 314, 317
 pwm_d9_dev (C macro), 314, 317–319, 321, 322, 324
 pwm_device (C++ member), 155
 PWM_DEVICE_MAX (C macro), 333, 334, 336
 pwm_get_duty (C++ function), 155
 pwm_init (C++ function), 154
 pwm_pin_to_device (C++ function), 155
 pwm_set_duty (C++ function), 154

Q

queue (module), 196
 queue_buffer_t (C++ class), 199
 queue_buffer_t::begin_p (C++ member), 199
 queue_buffer_t::end_p (C++ member), 199
 queue_buffer_t::read_p (C++ member), 199
 queue_buffer_t::size (C++ member), 199
 queue_buffer_t::write_p (C++ member), 199
 queue_init (C++ function), 198
 QUEUE_INIT_DECL (C macro), 197
 queue_read (C++ function), 198
 queue_size (C++ function), 199
 queue_start (C++ function), 198
 QUEUE_STATE_INITIALIZED (C++ class), 197
 QUEUE_STATE_RUNNING (C++ class), 197
 QUEUE_STATE_STOPPED (C++ class), 197
 queue_state_t (C++ type), 197
 queue_stop (C++ function), 198
 queue_stop_isr (C++ function), 198
 queue_t (C++ class), 199
 queue_t::base (C++ member), 200
 queue_t::buf_p (C++ member), 200
 queue_t::buffer (C++ member), 200
 queue_t::left (C++ member), 200
 queue_t::size (C++ member), 200
 queue_t::state (C++ member), 200
 queue_unused_size (C++ function), 199
 queue_unused_size_isr (C++ function), 199
 queue_write (C++ function), 198
 queue_write_isr (C++ function), 199

R

re (module), 294
 re_compile (C++ function), 295
 RE_DOTALL (C macro), 295
 re_group_t (C++ class), 296

- re_group_t::buf_p (C++ member), 296
 - re_group_t::size (C++ member), 296
 - RE_IGNORECASE (C macro), 295
 - re_match (C++ function), 296
 - RE_MULTILINE (C macro), 295
 - REQUEST_GET_DESCRIPTOR (C macro), 169
 - REQUEST_GET_STATUS (C macro), 169
 - REQUEST_SET_ADDRESS (C macro), 169
 - REQUEST_SET_CONFIGURATION (C macro), 169
 - REQUEST_TYPE_DATA_DIRECTION_DEVICE_TO_HOST (C macro), 169
 - REQUEST_TYPE_DATA_DIRECTION_HOST_TO_DEVICE (C macro), 169
 - REQUEST_TYPE_DATA_MASK (C macro), 169
 - REQUEST_TYPE_RECIPIENT_DEVICE (C macro), 169
 - REQUEST_TYPE_RECIPIENT_ENDPOINT (C macro), 169
 - REQUEST_TYPE_RECIPIENT_INTERFACE (C macro), 169
 - REQUEST_TYPE_RECIPIENT_MASK (C macro), 169
 - REQUEST_TYPE_RECIPIENT_OTHER (C macro), 169
 - REQUEST_TYPE_TYPE_CLASS (C macro), 169
 - REQUEST_TYPE_TYPE_MASK (C macro), 169
 - REQUEST_TYPE_TYPE_STANDARD (C macro), 169
 - REQUEST_TYPE_TYPE_VENDOR (C macro), 169
 - rwlock (module), 200
 - rwlock_init (C++ function), 200
 - rwlock_module_init (C++ function), 200
 - rwlock_reader_give (C++ function), 201
 - rwlock_reader_give_isr (C++ function), 201
 - rwlock_reader_take (C++ function), 200
 - rwlock_t (C++ class), 201
 - rwlock_t::number_of_readers (C++ member), 201
 - rwlock_t::number_of_writers (C++ member), 201
 - rwlock_t::readers_p (C++ member), 201
 - rwlock_t::writers_p (C++ member), 201
 - rwlock_writer_give (C++ function), 201
 - rwlock_writer_give_isr (C++ function), 201
 - rwlock_writer_take (C++ function), 201
 - RXCIE0 (C macro), 335
 - RXEN0 (C macro), 335
- ## S
- sam3x8e (module), 336
 - SAM_PA (C macro), 337
 - SAM_PB (C macro), 337
 - SAM_PC (C macro), 337
 - SAM_PD (C macro), 337
 - sd (module), 155
 - SD_BLOCK_SIZE (C macro), 156
 - SD_C_SIZE (C macro), 156
 - SD_C_SIZE_MULT (C macro), 156
 - SD_CCC (C macro), 156
 - sd_cid_t (C++ class), 157
 - sd_cid_t::crc (C++ member), 157
 - sd_cid_t::mdt (C++ member), 157
 - sd_cid_t::mid (C++ member), 157
 - sd_cid_t::oid (C++ member), 157
 - sd_cid_t::pnm (C++ member), 157
 - sd_cid_t::prv (C++ member), 157
 - sd_cid_t::psn (C++ member), 157
 - SD_CSD_STRUCTURE_V1 (C macro), 156
 - SD_CSD_STRUCTURE_V2 (C macro), 156
 - sd_csd_t (C++ type), 160
 - sd_csd_t::v1 (C++ member), 160
 - sd_csd_t::v2 (C++ member), 160
 - sd_csd_v1_t (C++ class), 157
 - sd_csd_v1_t::c_size_high (C++ member), 158
 - sd_csd_v1_t::c_size_low (C++ member), 158
 - sd_csd_v1_t::c_size_mid (C++ member), 158
 - sd_csd_v1_t::c_size_mult_high (C++ member), 158
 - sd_csd_v1_t::c_size_mult_low (C++ member), 158
 - sd_csd_v1_t::ccc_high (C++ member), 158
 - sd_csd_v1_t::ccc_low (C++ member), 158
 - sd_csd_v1_t::copy (C++ member), 159
 - sd_csd_v1_t::crc (C++ member), 159
 - sd_csd_v1_t::csd_structure (C++ member), 158
 - sd_csd_v1_t::dsr_imp (C++ member), 158
 - sd_csd_v1_t::erase_blk_en (C++ member), 158
 - sd_csd_v1_t::file_format (C++ member), 158
 - sd_csd_v1_t::file_format_grp (C++ member), 159
 - sd_csd_v1_t::nsac (C++ member), 158
 - sd_csd_v1_t::perm_write_protect (C++ member), 159
 - sd_csd_v1_t::r2w_factor (C++ member), 158
 - sd_csd_v1_t::read_bl_len (C++ member), 158
 - sd_csd_v1_t::read_bl_partial (C++ member), 158
 - sd_csd_v1_t::read_blk_misalign (C++ member), 158
 - sd_csd_v1_t::reserved1 (C++ member), 158
 - sd_csd_v1_t::reserved2 (C++ member), 158
 - sd_csd_v1_t::reserved3 (C++ member), 158
 - sd_csd_v1_t::reserved4 (C++ member), 158
 - sd_csd_v1_t::reserved5 (C++ member), 158
 - sd_csd_v1_t::sector_size_high (C++ member), 158
 - sd_csd_v1_t::sector_size_low (C++ member), 158
 - sd_csd_v1_t::taac (C++ member), 158
 - sd_csd_v1_t::tmp_write_protect (C++ member), 159
 - sd_csd_v1_t::tran_speed (C++ member), 158
 - sd_csd_v1_t::vdd_r_curr_max (C++ member), 158
 - sd_csd_v1_t::vdd_r_curr_min (C++ member), 158
 - sd_csd_v1_t::vdd_w_curr_max (C++ member), 158
 - sd_csd_v1_t::vdd_w_curr_min (C++ member), 158
 - sd_csd_v1_t::wp_grp_enable (C++ member), 158
 - sd_csd_v1_t::wp_grp_size (C++ member), 158
 - sd_csd_v1_t::write_bl_len_high (C++ member), 158
 - sd_csd_v1_t::write_bl_len_low (C++ member), 158
 - sd_csd_v1_t::write_bl_partial (C++ member), 158

- sd_csd_v1_t::write_blk_misalign (C++ member), 158
- sd_csd_v2_t (C++ class), 159
- sd_csd_v2_t::c_size_high (C++ member), 159
- sd_csd_v2_t::c_size_low (C++ member), 159
- sd_csd_v2_t::c_size_mid (C++ member), 159
- sd_csd_v2_t::ccc_high (C++ member), 159
- sd_csd_v2_t::ccc_low (C++ member), 159
- sd_csd_v2_t::copy (C++ member), 160
- sd_csd_v2_t::crc (C++ member), 160
- sd_csd_v2_t::csd_structure (C++ member), 159
- sd_csd_v2_t::dsr_imp (C++ member), 159
- sd_csd_v2_t::erase_blk_en (C++ member), 159
- sd_csd_v2_t::file_format (C++ member), 160
- sd_csd_v2_t::file_format_grp (C++ member), 160
- sd_csd_v2_t::nsac (C++ member), 159
- sd_csd_v2_t::perm_write_protect (C++ member), 160
- sd_csd_v2_t::r2w_factor (C++ member), 159
- sd_csd_v2_t::read_bl_len (C++ member), 159
- sd_csd_v2_t::read_bl_partial (C++ member), 159
- sd_csd_v2_t::read_blk_misalign (C++ member), 159
- sd_csd_v2_t::reserved1 (C++ member), 159
- sd_csd_v2_t::reserved2 (C++ member), 159
- sd_csd_v2_t::reserved3 (C++ member), 159
- sd_csd_v2_t::reserved4 (C++ member), 159
- sd_csd_v2_t::reserved5 (C++ member), 159
- sd_csd_v2_t::reserved6 (C++ member), 159
- sd_csd_v2_t::reserved7 (C++ member), 160
- sd_csd_v2_t::sector_size_high (C++ member), 159
- sd_csd_v2_t::sector_size_low (C++ member), 159
- sd_csd_v2_t::taac (C++ member), 159
- sd_csd_v2_t::tmp_write_protect (C++ member), 160
- sd_csd_v2_t::tran_speed (C++ member), 159
- sd_csd_v2_t::wp_grp_enable (C++ member), 159
- sd_csd_v2_t::wp_grp_size (C++ member), 159
- sd_csd_v2_t::write_bl_len_high (C++ member), 159
- sd_csd_v2_t::write_bl_len_low (C++ member), 159
- sd_csd_v2_t::write_bl_partial (C++ member), 159
- sd_csd_v2_t::write_blk_misalign (C++ member), 159
- sd_driver_t (C++ class), 160
- sd_driver_t::spi_p (C++ member), 160
- sd_driver_t::type (C++ member), 160
- SD_ERR_CHECK_PATTERN (C macro), 155
- SD_ERR_CRC_ON_OFF (C macro), 155
- SD_ERR_GO_IDLE_STATE (C macro), 155
- SD_ERR_NORESPONSE_WAIT_FOR_DATA_START_BLOCK (C macro), 155
- SD_ERR_READ_COMMAND (C macro), 155
- SD_ERR_READ_DATA_START_BLOCK (C macro), 155
- SD_ERR_READ_OCR (C macro), 155
- SD_ERR_READ_WRONG_DATA_CRC (C macro), 155
- SD_ERR_SD_SEND_OP_COND (C macro), 155
- SD_ERR_SEND_IF_COND (C macro), 155
- SD_ERR_WRITE_BLOCK (C macro), 155
- SD_ERR_WRITE_BLOCK_SEND_STATUS (C macro), 156
- SD_ERR_WRITE_BLOCK_TOKEN_DATA_RES_ACCEPTED (C macro), 156
- SD_ERR_WRITE_BLOCK_WAIT_NOT_BUSY (C macro), 156
- sd_init (C++ function), 156
- sd_read_block (C++ function), 157
- sd_read_cid (C++ function), 156
- sd_read_csd (C++ function), 156
- SD_SECTOR_SIZE (C macro), 156
- sd_start (C++ function), 156
- sd_stop (C++ function), 156
- SD_WRITE_BL_LEN (C macro), 156
- sd_write_block (C++ function), 157
- sdio (module), 160
- sdio_0_dev (C macro), 328
- sdio_device (C++ member), 163
- SDIO_DEVICE_MAX (C macro), 337
- sdio_init (C++ function), 160
- sdio_io_read_direct (C++ function), 161
- sdio_io_read_extended (C++ function), 162
- SDIO_IO_RW_EXTENDED_BLOCK_MODE_BLOCK (C macro), 160
- SDIO_IO_RW_EXTENDED_BLOCK_MODE_BYTE (C macro), 160
- SDIO_IO_RW_EXTENDED_OP_CODE_FIXED_ADDRESS (C macro), 160
- SDIO_IO_RW_EXTENDED_OP_CODE_INCREMENTING_ADDRESS (C macro), 160
- sdio_io_rw_extended_t (C++ class), 163
- sdio_io_rw_extended_t::block_mode (C++ member), 163
- sdio_io_rw_extended_t::byte_block_count_7_0 (C++ member), 163
- sdio_io_rw_extended_t::byte_block_count_8 (C++ member), 163
- sdio_io_rw_extended_t::function_number (C++ member), 163
- sdio_io_rw_extended_t::op_code (C++ member), 163
- sdio_io_rw_extended_t::register_address_14_7 (C++ member), 163
- sdio_io_rw_extended_t::register_address_16_15 (C++ member), 163
- sdio_io_rw_extended_t::register_address_6_0 (C++ member), 163
- sdio_io_rw_extended_t::rw_flag (C++ member), 163
- sdio_io_send_op_cond (C++ function), 161
- sdio_io_write_direct (C++ function), 162
- sdio_io_write_extended (C++ function), 162
- sdio_module_init (C++ function), 160
- sdio_select_deselect_card (C++ function), 161
- sdio_send_relative_addr (C++ function), 161
- sdio_start (C++ function), 161
- sdio_stop (C++ function), 161

sem (module), 202
sem_give (C++ function), 203
sem_give_isr (C++ function), 203
sem_init (C++ function), 202
SEM_INIT_DECL (C macro), 202
sem_module_init (C++ function), 202
sem_t (C++ class), 203
sem_t::count (C++ member), 203
sem_t::count_max (C++ member), 203
sem_t::head_p (C++ member), 203
sem_take (C++ function), 202
service (module), 265
SERVICE_CONTROL_EVENT_START (C macro), 266
SERVICE_CONTROL_EVENT_STOP (C macro), 266
service_deregister (C++ function), 267
service_get_status_cb_t (C++ type), 266
service_init (C++ function), 266
service_module_init (C++ function), 266
service_register (C++ function), 267
service_start (C++ function), 266
service_status_running_t (C++ class), 266
service_status_stopped_t (C++ class), 266
service_status_t (C++ type), 266
service_stop (C++ function), 266
service_t (C++ class), 267
service_t::control (C++ member), 267
service_t::name_p (C++ member), 267
service_t::next_p (C++ member), 267
service_t::status_cb (C++ member), 267
setting_t (C++ class), 271
setting_t::address (C++ member), 271
setting_t::size (C++ member), 271
setting_t::type (C++ member), 271
setting_type_int16_t (C++ class), 269
setting_type_int32_t (C++ class), 269
setting_type_int8_t (C++ class), 269
setting_type_string_t (C++ class), 269
setting_type_t (C++ type), 269
settings (module), 267
SETTINGS_AREA_CRC_OFFSET (C macro), 269
settings_module_init (C++ function), 270
settings_read (C++ function), 270
settings_read_by_name (C++ function), 270
settings_reset (C++ function), 271
settings_write (C++ function), 270
settings_write_by_name (C++ function), 270
sha1 (module), 305
sha1_digest (C++ function), 306
sha1_init (C++ function), 306
sha1_t (C++ class), 306
sha1_t::buf (C++ member), 306
sha1_t::h (C++ member), 306
sha1_t::size (C++ member), 306
sha1_update (C++ function), 306
shell (module), 271
shell_history_elem_t (C++ class), 272
shell_history_elem_t::buf (C++ member), 273
shell_history_elem_t::next_p (C++ member), 273
shell_history_elem_t::prev_p (C++ member), 273
shell_init (C++ function), 272
shell_line_t (C++ class), 273
shell_line_t::buf (C++ member), 273
shell_line_t::cursor (C++ member), 273
shell_line_t::length (C++ member), 273
shell_main (C++ function), 272
shell_module_init (C++ function), 272
shell_t (C++ class), 273
shell_t::arg_p (C++ member), 273
shell_t::authorized (C++ member), 273
shell_t::buf (C++ member), 273
shell_t::carriage_return_received (C++ member), 273
shell_t::chin_p (C++ member), 273
shell_t::chout_p (C++ member), 273
shell_t::current_p (C++ member), 273
shell_t::head_p (C++ member), 273
shell_t::heap (C++ member), 273
shell_t::line (C++ member), 273
shell_t::line_valid (C++ member), 273
shell_t::match (C++ member), 273
shell_t::name_p (C++ member), 273
shell_t::newline_received (C++ member), 273
shell_t::password_p (C++ member), 273
shell_t::pattern (C++ member), 273
shell_t::prev_line (C++ member), 273
shell_t::tail_p (C++ member), 273
shell_t::username_p (C++ member), 273
socket (module), 259
socket_accept (C++ function), 262
socket_bind (C++ function), 261
socket_close (C++ function), 261
socket_connect (C++ function), 261
socket_connect_by_hostname (C++ function), 261
SOCKET_DOMAIN_INET (C macro), 260
socket_listen (C++ function), 261
socket_module_init (C++ function), 260
socket_open (C++ function), 261
socket_open_raw (C++ function), 260
socket_open_tcp (C++ function), 260
socket_open_udp (C++ function), 260
SOCKET_PROTO_ICMP (C macro), 260
socket_read (C++ function), 263
socket_recvfrom (C++ function), 262
socket_sendto (C++ function), 262
socket_size (C++ function), 263
socket_t (C++ class), 263
socket_t::args_p (C++ member), 264
socket_t::base (C++ member), 263
socket_t::closed (C++ member), 263

- socket_t::left (C++ member), 263
- socket_t::pbuf_p (C++ member), 263
- socket_t::pcb_p (C++ member), 263, 264
- socket_t::remote_addr (C++ member), 263
- socket_t::state (C++ member), 264
- socket_t::thrd_p (C++ member), 264
- socket_t::type (C++ member), 263
- SOCKET_TYPE_DGRAM (C macro), 260
- SOCKET_TYPE_RAW (C macro), 260
- SOCKET_TYPE_STREAM (C macro), 260
- socket_write (C++ function), 263
- spi (module), 163
- spi_0_dev (C macro), 330, 333
- spi_1_dev (C macro), 331, 333
- spi_2_dev (C macro), 331, 333
- spi_deselect (C++ function), 165
- spi_device (C++ member), 166
- SPI_DEVICE_MAX (C macro), 333–338
- spi_get (C++ function), 165
- spi_give_bus (C++ function), 164
- spi_h_dev (C macro), 327
- spi_init (C++ function), 164
- SPI_MODE_MASTER (C macro), 163
- SPI_MODE_SLAVE (C macro), 163
- spi_module_init (C++ function), 163
- spi_put (C++ function), 166
- spi_read (C++ function), 165
- spi_select (C++ function), 164
- SPI_SPEED_125KBPS (C macro), 163
- SPI_SPEED_1MBPS (C macro), 163
- SPI_SPEED_250KBPS (C macro), 163
- SPI_SPEED_2MBPS (C macro), 163
- SPI_SPEED_4MBPS (C macro), 163
- SPI_SPEED_500KBPS (C macro), 163
- SPI_SPEED_8MBPS (C macro), 163
- spi_start (C++ function), 164
- spi_stop (C++ function), 164
- spi_take_bus (C++ function), 164
- spi_transfer (C++ function), 165
- spi_v_dev (C macro), 327
- spi_write (C++ function), 165
- spiffs (C++ type), 230
- spiffs (module), 227
- SPIFFS_APPEND (C macro), 228
- spiffs_block_ix (C++ type), 230
- SPIFFS_CACHE_DBG (C macro), 228
- SPIFFS_CB_CREATED (C++ class), 231
- SPIFFS_CB_DELETED (C++ class), 231
- SPIFFS_CB_UPDATED (C++ class), 231
- spiffs_check (C++ function), 235
- spiffs_check_callback (C++ type), 230
- spiffs_check_callback_t (C++ type), 229
- SPIFFS_CHECK_DBG (C macro), 228
- SPIFFS_CHECK_DELETE_BAD_FILE (C++ class), 231
- SPIFFS_CHECK_DELETE_ORPHANED_INDEX (C++ class), 230
- SPIFFS_CHECK_DELETE_PAGE (C++ class), 230
- SPIFFS_CHECK_ERROR (C++ class), 230
- SPIFFS_CHECK_FIX_INDEX (C++ class), 230
- SPIFFS_CHECK_FIX_LOOKUP (C++ class), 230
- SPIFFS_CHECK_INDEX (C++ class), 230
- SPIFFS_CHECK_LOOKUP (C++ class), 230
- SPIFFS_CHECK_PAGE (C++ class), 230
- SPIFFS_CHECK_PROGRESS (C++ class), 230
- spiffs_check_report_t (C++ type), 230
- spiffs_check_type_t (C++ type), 230
- spiffs_clearerr (C++ function), 235
- spiffs_close (C++ function), 234
- spiffs_closedir (C++ function), 235
- spiffs_config (C++ type), 230
- spiffs_config_t (C++ class), 238
- spiffs_config_t::hal_erase_f (C++ member), 238
- spiffs_config_t::hal_read_f (C++ member), 238
- spiffs_config_t::hal_write_f (C++ member), 238
- spiffs_config_t::log_block_size (C++ member), 238
- spiffs_config_t::log_page_size (C++ member), 238
- spiffs_config_t::phys_addr (C++ member), 238
- spiffs_config_t::phys_erase_block (C++ member), 238
- spiffs_config_t::phys_size (C++ member), 238
- SPIFFS_CREAT (C macro), 228
- spiffs_creat (C++ function), 231
- SPIFFS_DBG (C macro), 228
- spiffs_DIR (C++ type), 230
- spiffs_dir_t (C++ class), 240
- spiffs_dir_t::block (C++ member), 240
- spiffs_dir_t::entry (C++ member), 240
- spiffs_dir_t::fs (C++ member), 240
- SPIFFS_DIRECT (C macro), 229
- spiffs_dirent (C++ type), 230
- spiffs_dirent_t (C++ class), 239
- spiffs_dirent_t::name (C++ member), 239
- spiffs_dirent_t::obj_id (C++ member), 239
- spiffs_dirent_t::pix (C++ member), 240
- spiffs_dirent_t::size (C++ member), 240
- spiffs_dirent_t::type (C++ member), 239
- spiffs_eof (C++ function), 237
- spiffs_erase_cb_t (C++ type), 229
- SPIFFS_ERR_BAD_DESCRIPTOR (C macro), 227
- SPIFFS_ERR_CONFLICTING_NAME (C macro), 228
- SPIFFS_ERR_DATA_SPAN_MISMATCH (C macro), 227
- SPIFFS_ERR_DELETED (C macro), 227
- SPIFFS_ERR_END_OF_OBJECT (C macro), 227
- SPIFFS_ERR_ERASE_FAIL (C macro), 228
- SPIFFS_ERR_FILE_CLOSED (C macro), 227
- SPIFFS_ERR_FILE_DELETED (C macro), 227

SPIFFS_ERR_FILE_EXISTS (C macro), 228
 SPIFFS_ERR_FULL (C macro), 227
 SPIFFS_ERR_INDEX_FREE (C macro), 228
 SPIFFS_ERR_INDEX_INVALID (C macro), 228
 SPIFFS_ERR_INDEX_LU (C macro), 228
 SPIFFS_ERR_INDEX_REF_FREE (C macro), 227
 SPIFFS_ERR_INDEX_REF_INVALID (C macro), 228
 SPIFFS_ERR_INDEX_REF_LU (C macro), 227
 SPIFFS_ERR_INDEX_SPAN_MISMATCH (C macro), 227
 SPIFFS_ERR_INTERNAL (C macro), 228
 SPIFFS_ERR_IS_FREE (C macro), 227
 SPIFFS_ERR_IS_INDEX (C macro), 227
 SPIFFS_ERR_MAGIC_NOT_POSSIBLE (C macro), 228
 SPIFFS_ERR_MOUNTED (C macro), 228
 SPIFFS_ERR_NAME_TOO_LONG (C macro), 228
 SPIFFS_ERR_NO_DELETED_BLOCKS (C macro), 228
 SPIFFS_ERR_NOT_A_FILE (C macro), 228
 SPIFFS_ERR_NOT_A_FS (C macro), 228
 SPIFFS_ERR_NOT_CONFIGURED (C macro), 228
 SPIFFS_ERR_NOT_FINALIZED (C macro), 227
 SPIFFS_ERR_NOT_FOUND (C macro), 227
 SPIFFS_ERR_NOT_INDEX (C macro), 227
 SPIFFS_ERR_NOT_MOUNTED (C macro), 227
 SPIFFS_ERR_NOT_READABLE (C macro), 228
 SPIFFS_ERR_NOT_WRITABLE (C macro), 228
 SPIFFS_ERR_OUT_OF_FILE_DESCS (C macro), 227
 SPIFFS_ERR_PROBE_NOT_A_FS (C macro), 228
 SPIFFS_ERR_PROBE_TOO_FEW_BLOCKS (C macro), 228
 SPIFFS_ERR_RO_ABORTED_OPERATION (C macro), 228
 SPIFFS_ERR_RO_NOT_IMPL (C macro), 228
 SPIFFS_ERR_TEST (C macro), 228
 spiffs_errno (C++ function), 235
 SPIFFS_EXCL (C macro), 229
 spiffs_fflush (C++ function), 234
 spiffs_file (C++ type), 230
 spiffs_file_callback (C++ type), 230
 spiffs_file_callback_t (C++ type), 230
 spiffs_file_t (C++ type), 229
 spiffs_fileop_type (C++ type), 230
 spiffs_fileop_type_t (C++ type), 231
 spiffs_flags (C++ type), 230
 spiffs_flags_t (C++ type), 229
 spiffs_format (C++ function), 236
 spiffs_fremove (C++ function), 233
 spiffs_fstat (C++ function), 234
 spiffs_gc (C++ function), 237
 SPIFFS_GC_DBG (C macro), 228
 spiffs_gc_quick (C++ function), 236
 spiffs_info (C++ function), 236
 SPIFFS_LOCK (C macro), 229
 spiffs_lseek (C++ function), 233
 spiffs_mode (C++ type), 230
 spiffs_mode_t (C++ type), 229
 spiffs_mount (C++ function), 231
 spiffs_mounted (C++ function), 236
 SPIFFS_O_APPEND (C macro), 228
 SPIFFS_O_CREAT (C macro), 228
 SPIFFS_O_DIRECT (C macro), 229
 SPIFFS_O_EXCL (C macro), 229
 SPIFFS_O_RDONLY (C macro), 229
 SPIFFS_O_RDWR (C macro), 229
 SPIFFS_O_TRUNC (C macro), 228
 SPIFFS_O_WRONLY (C macro), 229
 spiffs_obj_id (C++ type), 230
 spiffs_obj_type (C++ type), 230
 spiffs_obj_type_t (C++ type), 229
 SPIFFS_OK (C macro), 227
 spiffs_open (C++ function), 232
 spiffs_open_by_dirent (C++ function), 232
 spiffs_open_by_page (C++ function), 232
 spiffs_opendir (C++ function), 235
 spiffs_page_ix (C++ type), 230
 SPIFFS_RDONLY (C macro), 228
 SPIFFS_RDWR (C macro), 229
 spiffs_read (C++ function), 232
 spiffs_read_cb_t (C++ type), 229
 spiffs_readdir (C++ function), 235
 spiffs_remove (C++ function), 233
 spiffs_rename (C++ function), 234
 SPIFFS_SEEK_CUR (C macro), 229
 SPIFFS_SEEK_END (C macro), 229
 SPIFFS_SEEK_SET (C macro), 229
 spiffs_set_file_callback_func (C++ function), 237
 spiffs_span_ix (C++ type), 230
 spiffs_stat (C++ function), 234
 spiffs_stat_t (C++ class), 239
 spiffs_stat_t::name (C++ member), 239
 spiffs_stat_t::obj_id (C++ member), 239
 spiffs_stat_t::pix (C++ member), 239
 spiffs_stat_t::size (C++ member), 239
 spiffs_stat_t::type (C++ member), 239
 spiffs_t (C++ class), 238
 spiffs_t::block_count (C++ member), 238
 spiffs_t::cfg (C++ member), 238
 spiffs_t::check_cb_f (C++ member), 239
 spiffs_t::cleaning (C++ member), 239
 spiffs_t::config_magic (C++ member), 239
 spiffs_t::cursor_block_ix (C++ member), 238
 spiffs_t::cursor_obj_lu_entry (C++ member), 238
 spiffs_t::err_code (C++ member), 239
 spiffs_t::fd_count (C++ member), 239
 spiffs_t::fd_space (C++ member), 238
 spiffs_t::file_cb_f (C++ member), 239

spiffs_t::free_blocks (C++ member), 239
 spiffs_t::free_cursor_block_ix (C++ member), 238
 spiffs_t::free_cursor_obj_lu_entry (C++ member), 238
 spiffs_t::lu_work (C++ member), 238
 spiffs_t::max_erase_count (C++ member), 239
 spiffs_t::mounted (C++ member), 239
 spiffs_t::stats_p_allocated (C++ member), 239
 spiffs_t::stats_p_deleted (C++ member), 239
 spiffs_t::user_data (C++ member), 239
 spiffs_t::work (C++ member), 238
 spiffs_tell (C++ function), 237
 SPIFFS_TRUNC (C macro), 228
 SPIFFS_TYPE_DIR (C macro), 229
 SPIFFS_TYPE_FILE (C macro), 229
 SPIFFS_TYPE_HARD_LINK (C macro), 229
 SPIFFS_TYPE_SOFT_LINK (C macro), 229
 SPIFFS_UNLOCK (C macro), 229
 spiffs_unmount (C++ function), 231
 spiffs_write (C++ function), 233
 spiffs_write_cb_t (C++ type), 229
 SPIFFS_WRONLY (C macro), 229
 st2t (C++ function), 109
 std (module), 296
 std_module_init (C++ function), 296
 std_printf (C++ function), 297
 STD_PRINTF_DEBUG (C macro), 122
 std_strip (C++ function), 299
 std_strtol (C++ function), 298
 stm32f100rb (module), 337
 stm32f205rg (module), 337
 stm32f303vc (module), 337
 stm32f3discovery (module), 328
 stm32vldiscovery (module), 331
 STRINGIFY (C macro), 121
 STRINGIFY2 (C macro), 121
 sys (C++ member), 111
 sys (module), 108
 sys_get_config (C++ function), 111
 sys_get_info (C++ function), 111
 sys_get_stdin (C++ function), 110
 sys_get_stdout (C++ function), 110
 sys_interrupt_cpu_usage_get (C++ function), 111
 sys_interrupt_cpu_usage_reset (C++ function), 111
 sys_lock (C++ function), 110
 sys_lock_isr (C++ function), 110
 sys_module_init (C++ function), 109
 sys_reboot (C++ function), 109
 sys_set_on_fatal_callback (C++ function), 109
 sys_set_stdin (C++ function), 110
 sys_set_stdout (C++ function), 110
 sys_start (C++ function), 109
 sys_stop (C++ function), 109
 sys_t (C++ class), 111
 sys_t::on_fatal_callback (C++ member), 111

sys_t::start (C++ member), 111
 sys_t::stdin_p (C++ member), 111
 sys_t::stdout_p (C++ member), 111
 sys_t::tick (C++ member), 111
 sys_t::time (C++ member), 111
 SYS_TICK_MAX (C macro), 109
 sys_tick_t (C++ type), 109
 sys_unlock (C++ function), 110
 sys_unlock_isr (C++ function), 110

T

t2st (C++ function), 109
 thrd (module), 111
 THRD_CONTEXT_LOAD_ISR (C macro), 113
 THRD_CONTEXT_STORE_ISR (C macro), 113
 thrd_environment_t (C++ class), 117
 thrd_environment_t::max_number_of_variables (C++ member), 117
 thrd_environment_t::number_of_variables (C++ member), 117
 thrd_environment_t::variables_p (C++ member), 117
 thrd_environment_variable_t (C++ class), 117
 thrd_environment_variable_t::name_p (C++ member), 117
 thrd_environment_variable_t::value_p (C++ member), 117
 thrd_get_by_name (C++ function), 115
 thrd_get_env (C++ function), 116
 thrd_get_global_env (C++ function), 116
 thrd_get_log_mask (C++ function), 115
 thrd_get_name (C++ function), 115
 thrd_get_prio (C++ function), 115
 thrd_init_env (C++ function), 116
 thrd_init_global_env (C++ function), 115
 thrd_join (C++ function), 114
 thrd_module_init (C++ function), 113
 THRD_RESCHEDULE_ISR (C macro), 113
 thrd_resume (C++ function), 113
 thrd_resume_isr (C++ function), 117
 thrd_self (C++ function), 114
 thrd_set_env (C++ function), 116
 thrd_set_global_env (C++ function), 115
 thrd_set_log_mask (C++ function), 115
 thrd_set_name (C++ function), 114
 thrd_set_prio (C++ function), 115
 thrd_sleep (C++ function), 114
 thrd_sleep_ms (C++ function), 114
 thrd_sleep_us (C++ function), 114
 thrd_spawn (C++ function), 113
 THRD_STACK (C macro), 113
 thrd_suspend (C++ function), 113
 thrd_suspend_isr (C++ function), 116
 thrd_t (C++ class), 117
 thrd_t::err (C++ member), 117

thrd_t::log_mask (C++ member), 117
thrd_t::name_p (C++ member), 117
thrd_t::next_p (C++ member), 117
thrd_t::port (C++ member), 117
thrd_t::prev_p (C++ member), 117
thrd_t::prio (C++ member), 117
thrd_t::state (C++ member), 117
thrd_t::timer_p (C++ member), 117
thrd_yield (C++ function), 114
thrd_yield_isr (C++ function), 117
time (module), 118
time_busy_wait_us (C++ function), 118
time_diff (C++ function), 118
time_get (C++ function), 118
time_set (C++ function), 118
time_t (C++ class), 119
time_t::nanoseconds (C++ member), 119
time_t::seconds (C++ member), 119
time_unix_time_to_date (C++ function), 118
timer (module), 119
timer_callback_t (C++ type), 120
timer_init (C++ function), 120
timer_module_init (C++ function), 120
TIMER_PERIODIC (C macro), 120
timer_start (C++ function), 120
timer_start_isr (C++ function), 120
timer_stop (C++ function), 121
timer_stop_isr (C++ function), 121
timer_t (C++ class), 121
timer_t::arg_p (C++ member), 121
timer_t::callback (C++ member), 121
timer_t::delta (C++ member), 121
timer_t::flags (C++ member), 121
timer_t::next_p (C++ member), 121
timer_t::timeout (C++ member), 121
TOKENPASTE (C macro), 121
TOKENPASTE2 (C macro), 121
TXC0 (C macro), 335
TXCIE0 (C macro), 335
TXEN0 (C macro), 335
types (module), 121

U

U2X0 (C macro), 335
uart (module), 166
uart_0_dev (C macro), 330, 333
uart_1_dev (C macro), 330, 333
uart_2_dev (C macro), 330, 333
uart_device (C++ member), 168
UART_DEVICE_MAX (C macro), 333–338
uart_init (C++ function), 167
uart_module_init (C++ function), 167
uart_read (C macro), 166
uart_rx_filter_cb_t (C++ type), 167

uart_set_rx_filter_cb (C++ function), 167
uart_soft (module), 168
uart_soft_driver_t (C++ class), 169
uart_soft_driver_t::baudrate (C++ member), 169
uart_soft_driver_t::chin (C++ member), 169
uart_soft_driver_t::chout (C++ member), 169
uart_soft_driver_t::rx_exti (C++ member), 169
uart_soft_driver_t::rx_pin (C++ member), 169
uart_soft_driver_t::sample_time (C++ member), 169
uart_soft_driver_t::tx_pin (C++ member), 169
uart_soft_init (C++ function), 168
uart_soft_read (C macro), 168
uart_soft_write (C macro), 168
uart_start (C++ function), 167
uart_stop (C++ function), 167
uart_write (C macro), 166
UCSZ00 (C macro), 334
UCSZ01 (C macro), 335
UCSZ02 (C macro), 335
UDRE0 (C macro), 335
UDRIE0 (C macro), 335
UNIQUE (C macro), 121
UNUSED (C macro), 121
UPE0 (C macro), 335
UPM00 (C macro), 335
UPM01 (C macro), 335
USART0_RX_vect (C macro), 334
USART0_TX_vect (C macro), 334
USART0_UDRE_vect (C macro), 334
usb (module), 169
USB_CDC_CONTROL_LINE_STATE (C macro), 171
USB_CDC_LINE_CODING (C macro), 171
usb_cdc_line_info_t (C++ class), 176
usb_cdc_line_info_t::char_format (C++ member), 176
usb_cdc_line_info_t::data_bits (C++ member), 176
usb_cdc_line_info_t::dte_rate (C++ member), 176
usb_cdc_line_info_t::parity_type (C++ member), 176
USB_CDC_SEND_BREAK (C macro), 171
USB_CLASS_APPLICATION_SPECIFIC (C macro), 170
USB_CLASS_AUDIO (C macro), 170
USB_CLASS_AUDIO_VIDEO_DEVICES (C macro), 170
USB_CLASS_BILLBOARD_DEVICE_CLASS (C macro), 170
USB_CLASS_CDC_CONTROL (C macro), 170
USB_CLASS_CDC_DATA (C macro), 170
USB_CLASS_CONTENT_SECURITY (C macro), 170
USB_CLASS_DIAGNOSTIC_DEVICE (C macro), 170
USB_CLASS_HID (C macro), 170
USB_CLASS_HID_PROTOCOL_KEYBOARD (C macro), 181
USB_CLASS_HID_PROTOCOL_MOUSE (C macro), 181

- USB_CLASS_HID_PROTOCOL_NONE (C macro), 181
- USB_CLASS_HID_SUBCLASS_BOOT_INTERFACE (C macro), 181
- USB_CLASS_HID_SUBCLASS_NONE (C macro), 181
- USB_CLASS_HUB (C macro), 170
- USB_CLASS_IMAGE (C macro), 170
- USB_CLASS_MASS_STORAGE (C macro), 170
- USB_CLASS_MISCELLANEOUS (C macro), 170
- USB_CLASS_PERSONAL_HEALTHCARE (C macro), 170
- USB_CLASS_PHYSICAL (C macro), 170
- USB_CLASS_PRINTER (C macro), 170
- USB_CLASS_SMART_CARD (C macro), 170
- USB_CLASS_USE_INTERFACE (C macro), 170
- USB_CLASS_VENDOR_SPECIFIC (C macro), 170
- USB_CLASS_VIDEO (C macro), 170
- USB_CLASS_WIRELESS_CONTROLLER (C macro), 170
- usb_desc_get_class (C++ function), 172
- usb_desc_get_configuration (C++ function), 171
- usb_desc_get_endpoint (C++ function), 171
- usb_desc_get_interface (C++ function), 171
- usb_descriptor_cdc_acm_t (C++ class), 175
- usb_descriptor_cdc_acm_t::capabilities (C++ member), 175
- usb_descriptor_cdc_acm_t::descriptor_type (C++ member), 175
- usb_descriptor_cdc_acm_t::length (C++ member), 175
- usb_descriptor_cdc_acm_t::sub_type (C++ member), 175
- usb_descriptor_cdc_call_management_t (C++ class), 176
- usb_descriptor_cdc_call_management_t::capabilities (C++ member), 176
- usb_descriptor_cdc_call_management_t::data_interface (C++ member), 176
- usb_descriptor_cdc_call_management_t::descriptor_type (C++ member), 176
- usb_descriptor_cdc_call_management_t::length (C++ member), 176
- usb_descriptor_cdc_call_management_t::sub_type (C++ member), 176
- usb_descriptor_cdc_header_t (C++ class), 175
- usb_descriptor_cdc_header_t::bcd (C++ member), 175
- usb_descriptor_cdc_header_t::descriptor_type (C++ member), 175
- usb_descriptor_cdc_header_t::length (C++ member), 175
- usb_descriptor_cdc_header_t::sub_type (C++ member), 175
- usb_descriptor_cdc_union_t (C++ class), 175
- usb_descriptor_cdc_union_t::descriptor_type (C++ member), 175
- usb_descriptor_cdc_union_t::length (C++ member), 175
- usb_descriptor_cdc_union_t::master_interface (C++ member), 176
- usb_descriptor_cdc_union_t::slave_interface (C++ member), 176
- usb_descriptor_cdc_union_t::sub_type (C++ member), 175
- usb_descriptor_configuration_t (C++ class), 173
- usb_descriptor_configuration_t::configuration (C++ member), 174
- usb_descriptor_configuration_t::configuration_attributes (C++ member), 174
- usb_descriptor_configuration_t::configuration_value (C++ member), 174
- usb_descriptor_configuration_t::descriptor_type (C++ member), 174
- usb_descriptor_configuration_t::length (C++ member), 174
- usb_descriptor_configuration_t::max_power (C++ member), 174
- usb_descriptor_configuration_t::num_interfaces (C++ member), 174
- usb_descriptor_configuration_t::total_length (C++ member), 174
- usb_descriptor_device_t (C++ class), 173
- usb_descriptor_device_t::bcd_device (C++ member), 173
- usb_descriptor_device_t::bcd_usb (C++ member), 173
- usb_descriptor_device_t::descriptor_type (C++ member), 173
- usb_descriptor_device_t::device_class (C++ member), 173
- usb_descriptor_device_t::device_protocol (C++ member), 173
- usb_descriptor_device_t::device_subclass (C++ member), 173
- usb_descriptor_device_t::id_product (C++ member), 173
- usb_descriptor_device_t::id_vendor (C++ member), 173
- usb_descriptor_device_t::length (C++ member), 173
- usb_descriptor_device_t::manufacturer (C++ member), 173
- usb_descriptor_device_t::max_packet_size_0 (C++ member), 173
- usb_descriptor_device_t::num_configurations (C++ member), 173
- usb_descriptor_device_t::product (C++ member), 173
- usb_descriptor_device_t::serial_number (C++ member), 173
- usb_descriptor_endpoint_t (C++ class), 174
- usb_descriptor_endpoint_t::attributes (C++ member), 174
- usb_descriptor_endpoint_t::descriptor_type (C++ member), 174
- usb_descriptor_endpoint_t::endpoint_address (C++ member), 174
- usb_descriptor_endpoint_t::interval (C++ member), 174
- usb_descriptor_endpoint_t::length (C++ member), 174

[usb_descriptor_endpoint_t::max_packet_size \(C++ member\), 174](#)
[usb_descriptor_header_t \(C++ class\), 173](#)
[usb_descriptor_header_t::descriptor_type \(C++ member\), 173](#)
[usb_descriptor_header_t::length \(C++ member\), 173](#)
[usb_descriptor_interface_association_t \(C++ class\), 175](#)
[usb_descriptor_interface_association_t::descriptor_type \(C++ member\), 175](#)
[usb_descriptor_interface_association_t::first_interface \(C++ member\), 175](#)
[usb_descriptor_interface_association_t::function \(C++ member\), 175](#)
[usb_descriptor_interface_association_t::function_class \(C++ member\), 175](#)
[usb_descriptor_interface_association_t::function_protocol \(C++ member\), 175](#)
[usb_descriptor_interface_association_t::function_subclass \(C++ member\), 175](#)
[usb_descriptor_interface_association_t::interface_count \(C++ member\), 175](#)
[usb_descriptor_interface_association_t::length \(C++ member\), 175](#)
[usb_descriptor_interface_t \(C++ class\), 174](#)
[usb_descriptor_interface_t::alternate_setting \(C++ member\), 174](#)
[usb_descriptor_interface_t::descriptor_type \(C++ member\), 174](#)
[usb_descriptor_interface_t::interface \(C++ member\), 174](#)
[usb_descriptor_interface_t::interface_class \(C++ member\), 174](#)
[usb_descriptor_interface_t::interface_number \(C++ member\), 174](#)
[usb_descriptor_interface_t::interface_protocol \(C++ member\), 174](#)
[usb_descriptor_interface_t::interface_subclass \(C++ member\), 174](#)
[usb_descriptor_interface_t::length \(C++ member\), 174](#)
[usb_descriptor_interface_t::num_endpoints \(C++ member\), 174](#)
[usb_descriptor_string_t \(C++ class\), 174](#)
[usb_descriptor_string_t::descriptor_type \(C++ member\), 175](#)
[usb_descriptor_string_t::length \(C++ member\), 175](#)
[usb_descriptor_string_t::string \(C++ member\), 175](#)
[usb_descriptor_t \(C++ type\), 176](#)
[usb_descriptor_t::configuration \(C++ member\), 176](#)
[usb_descriptor_t::device \(C++ member\), 176](#)
[usb_descriptor_t::endpoint \(C++ member\), 176](#)
[usb_descriptor_t::header \(C++ member\), 176](#)
[usb_descriptor_t::interface \(C++ member\), 176](#)
[usb_descriptor_t::string \(C++ member\), 176](#)
[usb_device \(C++ member\), 172](#)
[usb_device \(module\), 177](#)
[usb_device_class_cdc \(module\), 177](#)
[usb_device_class_cdc_driver_t \(C++ class\), 178](#)
[usb_device_class_cdc_driver_t::base \(C++ member\), 179](#)
[usb_device_class_cdc_driver_t::chin \(C++ member\), 179](#)
[usb_device_class_cdc_driver_t::chout \(C++ member\), 179](#)
[usb_device_class_cdc_driver_t::control_interface \(C++ member\), 179](#)
[usb_device_class_cdc_driver_t::drv_p \(C++ member\), 179](#)
[usb_device_class_cdc_driver_t::endpoint_in \(C++ member\), 179](#)
[usb_device_class_cdc_driver_t::endpoint_out \(C++ member\), 179](#)
[usb_device_class_cdc_driver_t::line_info \(C++ member\), 179](#)
[usb_device_class_cdc_driver_t::line_state \(C++ member\), 179](#)
[usb_device_class_cdc_init \(C++ function\), 178](#)
[usb_device_class_cdc_input_isr \(C++ function\), 178](#)
[usb_device_class_cdc_is_connected \(C++ function\), 178](#)
[usb_device_class_cdc_module_init \(C++ function\), 178](#)
[usb_device_class_cdc_read \(C macro\), 177](#)
[usb_device_class_cdc_write \(C macro\), 178](#)
[USB_DEVICE_MAX \(C macro\), 334](#)
[usb_device_module_init \(C++ function\), 179](#)
[usb_device_read_isr \(C++ function\), 180](#)
[usb_device_start \(C++ function\), 179](#)
[usb_device_stop \(C++ function\), 179](#)
[usb_device_write \(C++ function\), 180](#)
[usb_device_write_isr \(C++ function\), 180](#)
[usb_format_descriptors \(C++ function\), 171](#)
[usb_host \(module\), 180](#)
[usb_host_class_hid \(module\), 181](#)
[usb_host_class_hid_device_t \(C++ class\), 182](#)
[usb_host_class_hid_device_t::buf \(C++ member\), 182](#)
[usb_host_class_hid_driver_t \(C++ class\), 182](#)
[usb_host_class_hid_driver_t::device_driver \(C++ member\), 182](#)
[usb_host_class_hid_driver_t::devices_p \(C++ member\), 182](#)
[usb_host_class_hid_driver_t::length \(C++ member\), 182](#)
[usb_host_class_hid_driver_t::size \(C++ member\), 182](#)
[usb_host_class_hid_driver_t::usb_p \(C++ member\), 182](#)
[usb_host_class_hid_init \(C++ function\), 181](#)
[usb_host_class_hid_start \(C++ function\), 181](#)
[usb_host_class_hid_stop \(C++ function\), 181](#)
[usb_host_class_mass_storage \(module\), 182](#)
[usb_host_class_mass_storage_device_read \(C++ function\), 182](#)
[usb_host_class_mass_storage_device_t \(C++ class\), 182](#)
[usb_host_class_mass_storage_device_t::buf \(C++ member\), 183](#)
[usb_host_class_mass_storage_driver_t \(C++ class\), 183](#)

[usb_host_class_mass_storage_driver_t::device_driver \(C++ member\), 183](#)
[usb_host_class_mass_storage_driver_t::devices_p \(C++ member\), 183](#)
[usb_host_class_mass_storage_driver_t::length \(C++ member\), 183](#)
[usb_host_class_mass_storage_driver_t::size \(C++ member\), 183](#)
[usb_host_class_mass_storage_driver_t::usb_p \(C++ member\), 183](#)
[usb_host_class_mass_storage_init \(C++ function\), 182](#)
[usb_host_class_mass_storage_start \(C++ function\), 182](#)
[usb_host_class_mass_storage_stop \(C++ function\), 182](#)
[usb_host_device_close \(C++ function\), 185](#)
[usb_host_device_control_transfer \(C++ function\), 185](#)
[usb_host_device_driver_t \(C++ class\), 186](#)
[usb_host_device_driver_t::enumerate \(C++ member\), 186](#)
[usb_host_device_driver_t::next_p \(C++ member\), 186](#)
[usb_host_device_driver_t::supports \(C++ member\), 186](#)
[usb_host_device_open \(C++ function\), 184](#)
[usb_host_device_read \(C++ function\), 185](#)
[usb_host_device_set_configuration \(C++ function\), 185](#)
[USB_HOST_DEVICE_STATE_ATTACHED \(C macro\), 183](#)
[USB_HOST_DEVICE_STATE_NONE \(C macro\), 183](#)
[usb_host_device_t \(C++ class\), 186](#)
[usb_host_device_t::address \(C++ member\), 186](#)
[usb_host_device_t::buf \(C++ member\), 186](#)
[usb_host_device_t::conf_p \(C++ member\), 186](#)
[usb_host_device_t::configuration \(C++ member\), 186](#)
[usb_host_device_t::description_p \(C++ member\), 186](#)
[usb_host_device_t::dev_p \(C++ member\), 186](#)
[usb_host_device_t::id \(C++ member\), 186](#)
[usb_host_device_t::max_packet_size \(C++ member\), 186](#)
[usb_host_device_t::pid \(C++ member\), 186](#)
[usb_host_device_t::pipes \(C++ member\), 186](#)
[usb_host_device_t::self_p \(C++ member\), 186](#)
[usb_host_device_t::size \(C++ member\), 186](#)
[usb_host_device_t::state \(C++ member\), 186](#)
[usb_host_device_t::vid \(C++ member\), 186](#)
[usb_host_device_write \(C++ function\), 185](#)
[usb_host_driver_add \(C++ function\), 184](#)
[usb_host_driver_remove \(C++ function\), 184](#)
[usb_host_init \(C++ function\), 183](#)
[usb_host_module_init \(C++ function\), 183](#)
[usb_host_start \(C++ function\), 184](#)
[usb_host_stop \(C++ function\), 184](#)
[usb_message_add_t \(C++ class\), 176](#)
[usb_message_add_t::device \(C++ member\), 177](#)
[usb_message_add_t::header \(C++ member\), 177](#)
[usb_message_header_t \(C++ class\), 176](#)
[usb_message_header_t::type \(C++ member\), 176](#)
[usb_message_t \(C++ type\), 177](#)
[usb_message_t::add \(C++ member\), 177](#)
[usb_message_t::header \(C++ member\), 177](#)
[USB_MESSAGE_TYPE_ADD \(C macro\), 171](#)
[USB_MESSAGE_TYPE_REMOVE \(C macro\), 171](#)
[USB_PIPE_TYPE_BULK \(C macro\), 183](#)
[USB_PIPE_TYPE_CONTROL \(C macro\), 183](#)
[USB_PIPE_TYPE_INTERRUPT \(C macro\), 183](#)
[USB_PIPE_TYPE_ISOCHRONOUS \(C macro\), 183](#)
[usb_setup_t \(C++ class\), 172](#)
[usb_setup_t::configuration_value \(C++ member\), 173](#)
[usb_setup_t::descriptor_index \(C++ member\), 172](#)
[usb_setup_t::descriptor_type \(C++ member\), 172](#)
[usb_setup_t::device_address \(C++ member\), 173](#)
[usb_setup_t::feature_selector \(C++ member\), 172](#)
[usb_setup_t::index \(C++ member\), 173](#)
[usb_setup_t::language_id \(C++ member\), 172](#)
[usb_setup_t::length \(C++ member\), 173](#)
[usb_setup_t::request \(C++ member\), 172](#)
[usb_setup_t::request_type \(C++ member\), 172](#)
[usb_setup_t::value \(C++ member\), 173](#)
[usb_setup_t::zero \(C++ member\), 173](#)
[usb_setup_t::zero0 \(C++ member\), 172](#)
[usb_setup_t::zero1 \(C++ member\), 172](#)
[usb_setup_t::zero_interface_endpoint \(C++ member\), 172](#)
[USBS0 \(C macro\), 335](#)

V

[VERSION_STR \(C macro\), 109](#)

W

[watchdog \(module\), 187](#)
[watchdog_kick \(C++ function\), 187](#)
[watchdog_module_init \(C++ function\), 187](#)
[watchdog_start_ms \(C++ function\), 187](#)
[watchdog_stop \(C++ function\), 187](#)