Introduction

The wESP32™ is an ESP32 core board with wired Ethernet connectivity and Power over Ethernet that was designed to help Makers quickly create zero-setup, single-wire-install connected devices. By combining a powerful microcontroller with excellent community support, 13 W of power, and reliable connectivity in a compact footprint, the wESP32™ gives you a head start in your next IoT design and allows you to focus on your application.

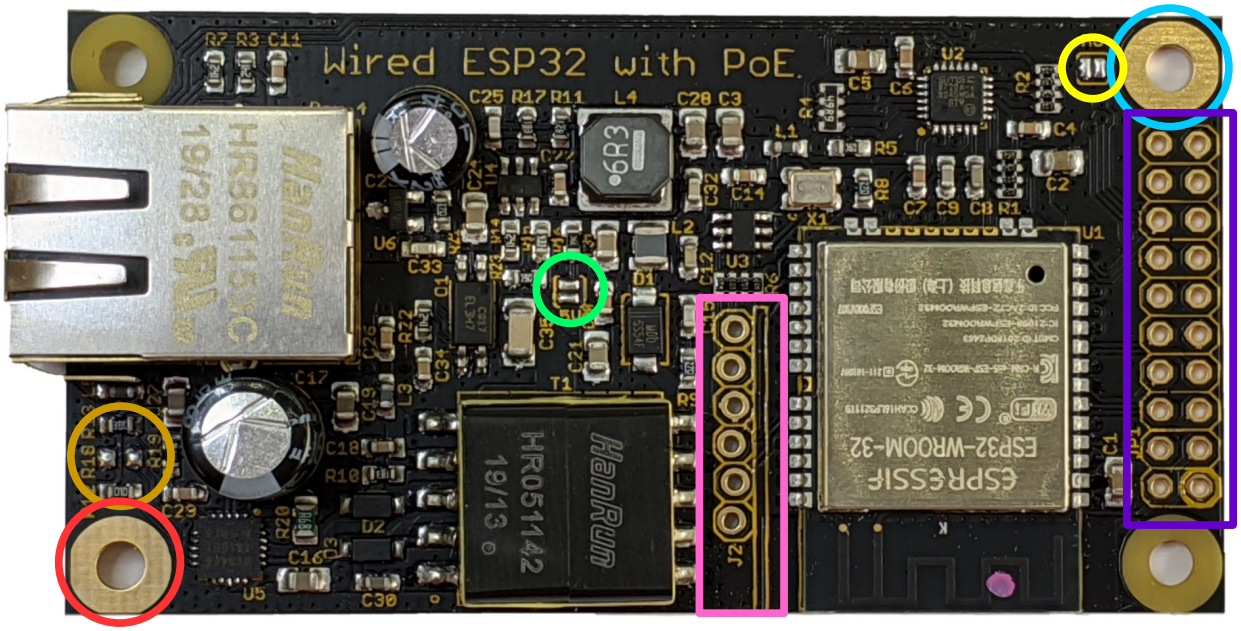
The wESP32™ provides the following functionality and features:

- Espressif Systems ESP32-WROOM-32 module with 4 MB of flash for revisions prior to 7, ESP32-WROOM-32E module with 16 MB of flash from revision 7 onward.
- Optional wESP32-Prog module for programming and/or USB serial console can be plugged in as needed or soldered in permanently.
- IEEE 802.3at Type 1 Class 0 compliant PoE with 12.95+ W of power available at the default 12 V output setting for V+.
- Optional 5 V output setting for V+ configurable by solder jumper with 5 W of power available.

- 3.3 V output with up to 6 W output power available (power taken from V+).
- Applying external power to V+ is supported, in case PoE is not available.
- Applying power through the wESP32-Prog USB port is supported (only provides USB 5 V on V+).
- High performance RJ-45 jack with optimized data path provides excellent data throughput.
- Compatible with PoE Mode A (power over data pairs) and PoE Mode B (power over spare pairs) and Auto-MDIX allowing the use of straight and cross-over Ethernet cables.
- Designed with full data and power path isolation to comply with the IEEE 802.3at 1500 Vrms isolation requirement.
- High quality PCB with 2 oz copper and mounting hole designed to sink heat away from the PoE PD controller for optimal thermal performance.
- 20-pin header footprint with 3.3 V, V+ power and 15 of the ESP32 GPIOs available for your application.
- Compact 75 mm by 40 mm footprint with 4 mounting holes compatible with common M2.5 or 4-40 screws.

Hardware

Physical specification



The image above shows the wESP32™ in detail with certain features highlighted.

The wESP32™ is constructed on a high-quality 1.2 mm thick FR4 PCB. The board is designed to provide electrical isolation between the ESP32 application portion and the Ethernet cable to ensure safety and performance as demanded by the IEEE 802.3at specification.

Mounting holes

Four 3 mm diameter mounting holes are located 3.5 mm from each board edge and are compatible with both M2.5 and 4-40 screws.

As can be seen in the image, two of the mounting holes are plated with metal and two are not. The mounting hole on the bottom left marked in red is electrically connected to the potential of the upstream PoE PSE (Power Sourcing Equipment) and is thermally coupled to the PoE PD (Powered Device) controller chip to the right of it. Aside of mounting, the main purpose of this hole is to be able to provide heat sinking for the PD controller when under high load by screwing a metal heat sink to the hole. Note that since this hole is electrically connected to the PSE, it should not be connected to a metal case the user can touch as this would negate the isolation.

The mounting hole on the top right marked in blue is by default not electrically connected to the rest of the board, but can be connected by closing solder jumper “HG” marked in yellow if the user wants to locally ground the system or electrically connect the ESP32 application side of the board to a metal case. As noted above, this mounting hole and the bottom left mounting hole should *not* be electrically connected as this would negate the isolation.

Classification resistor

On the left side of the board, marked in brown, is an unpopulated resistor footprint R18 that can be used to add an optional 0603 size power classification resistor. By default, with R18 not populated, the wESP32™ reports itself to the PoE PSE as a “Class 0” device, meaning classification is not implemented and up to 12.95 W is available to the device. The table below documents the resistor values that R18 can be populated with to indicate power requirement classification to the PSE:

R18 value	Power Class	Power range at PD	Description
> 681 Ω	0	0.44 – 12.95 W	Classification unimplemented
140 Ω 1%	1	0.44 – 3.84 W	Very low power
75 Ω 1%	2	3.84 – 6.49 W	Low power
48.7 Ω 1%	3	6.49 – 12.95 W	Mid power

5 V option

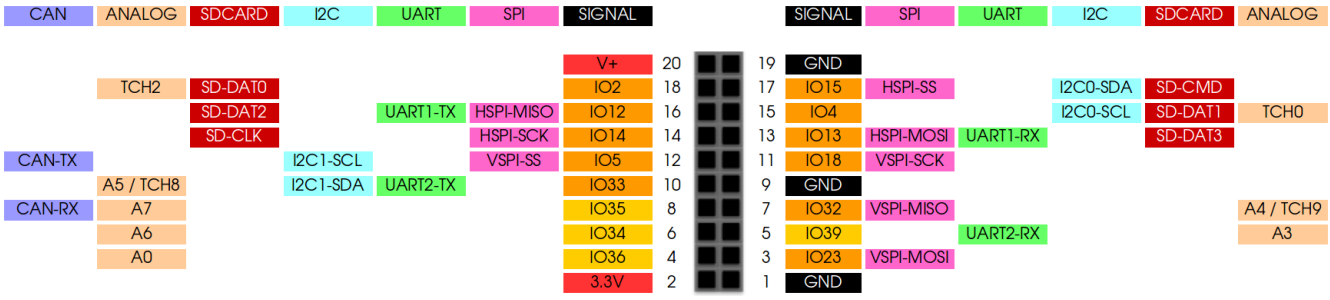
The feature marked in green in the middle of the board is the “5V” solder jumper. By default, the V+ power output has a voltage of 12 V. This configuration provides the maximum amount of output power capability (12.95+ W as specified for IEEE 802.3at Type 1 Class 0). However, sometimes the user may need 5 V instead. By bridging this solder jumper, the V+ power output voltage will be set to 5 V instead. In the 5 V output configuration, only 5 W of power is available.

GPIO header footprint

On the right side of the board, marked in purple blue, is the GPIO header footprint JP1. This is a bare through-hole 2 x 10 pin 0.1” footprint where the user can directly solder wires or add a header, for instance the optional 2 x 10 position female header. This is where the user can connect their application circuitry.

The central idea for the wESP32™ is that it provides processing, connectivity and power functionality, and the user can create a simple add-on board with their application specific sensors, level shifters, power drivers and connectors that plugs in to the GPIO header. The header is electrically connected to the ESP32 application side of the board and isolated from the Ethernet cable, so the user doesn’t need to worry about issues with touching or grounding anything connected to the GPIO header.

The wESP32™ makes many of the GPIO signals of the ESP32-WROOM-32 module available on the GPIO header, but some are missing because they are either used by the Ethernet MAC RMII interface or the programming circuitry. All pins that are on the GPIO header are exclusively available to the user application. The following graphic shows the available pins:



This documents the pinout of the GPIO header, and the default mapping of peripherals when using the Arduino development environment (and other environments with the concept of “board files” follow this layout). Please note that other development environments such as ESP32-IDF and MicroPython provide more flexibility, and allow many of these functions to be mapped to any available pin. However, if compatibility with future add-on boards that work with Arduino is desired, it may make

sense to stick to these mappings even for those development environments.

Notes:

- Not all pins have output capability. The input-only pins are marked in lighter orange.
- TCHx are touch channels, Ax are analog (ADC) inputs.
- UART0 is not mapped to the GPIO header but connected to programming / serial console port J2.
- Pin 1 is marked on the silk screen but when a header is installed it may not be possible to see this. In that case, pin 1 is the pin closest to the non-plated mounting hole, toward the edge of the board.
- Certain ESP32 pins are so-called “strapping pins” that configure the chip on boot. Of those, IO2, IO12 and IO15 are available on the GPIO header. The user is responsible for ensuring that the circuitry they attach does not interfere with the ESP32 boot or programming sequence by making sure their circuitry does not pull either IO2 or IO12 high when the system first is powered. There pins need to be either left floating (allowing the built-in pull-downs to do their job) or driven low on boot.

Programming / serial console footprint

In the middle, marked in pink, is the programming and/or serial console header footprint J2. Here the user can connect either their own ESP32 programmer or a wESP32-Prog module. The footprint uses a staggered layout so it is possible to temporarily press-fit a wESP32-Prog module into place for one time programming. For reliable operation during development it is recommended to actually solder the module permanently to the wESP32™, and of course if the application requires a USB serial connection, this can be another reason to install it permanently. The header is electrically connected to the ESP32 application side of the board and isolated from the Ethernet cable, so the user does not need to worry about connecting the USB to their PC while Ethernet is connected. The pinout of the header is documented below:

Pin	Name	Notes
1	IO0	Used to select boot mode for either normal boot (high or not driven since the wESP32™ contains a pull-up) or programming (low, see ESP32 documentation). Connect to programming tool DTR signal for auto programming by esptool.py. Note that after normal boot, this pad signal will be driven by a 50 MHz clock for the Ethernet PHY and MAC so do not drive it or put a load on it because it may interfere with the operation of the Ethernet subsystem.

2	EN	Used to reset the ESP32 and during programming. Connect to programming tool RTS signal for auto programming by esptool.py.
3	GND	Ground reference, connected to ESP32 application side of the board and isolated from Ethernet.
4	TXD	UART0 transmit line from the wESP32™, receive line as seen from the external programming / console device.
5	RXD	UART0 receive line from the wESP32™, transmit line as seen from the external programming / console device.
6	V+	Positive supply voltage. If no other power source is present, the wESP32™ can be powered from the connected programming / serial console device. Note that when connected to a PoE PSE, this pad will have 12 V power on it (or 5 V if the "5V" solder jumper is bridged), so make sure you have a blocking diode in your programming device or your device can handle this voltage if the pad is connected!

Pin 1 is toward the middle of the board, pin 6 near the edge of the board. An important thing to note is that to correctly operate, the levels of three other pins are important during boot:

- IO2 must be floating or driven low to correctly enter programming mode.
- IO12 must be floating or driven low to select the correct flash voltage during normal boot or when entering programming mode.
- IO15, if driven low during boot, suppresses boot messages printed by the ROM bootloader.

Revision changes

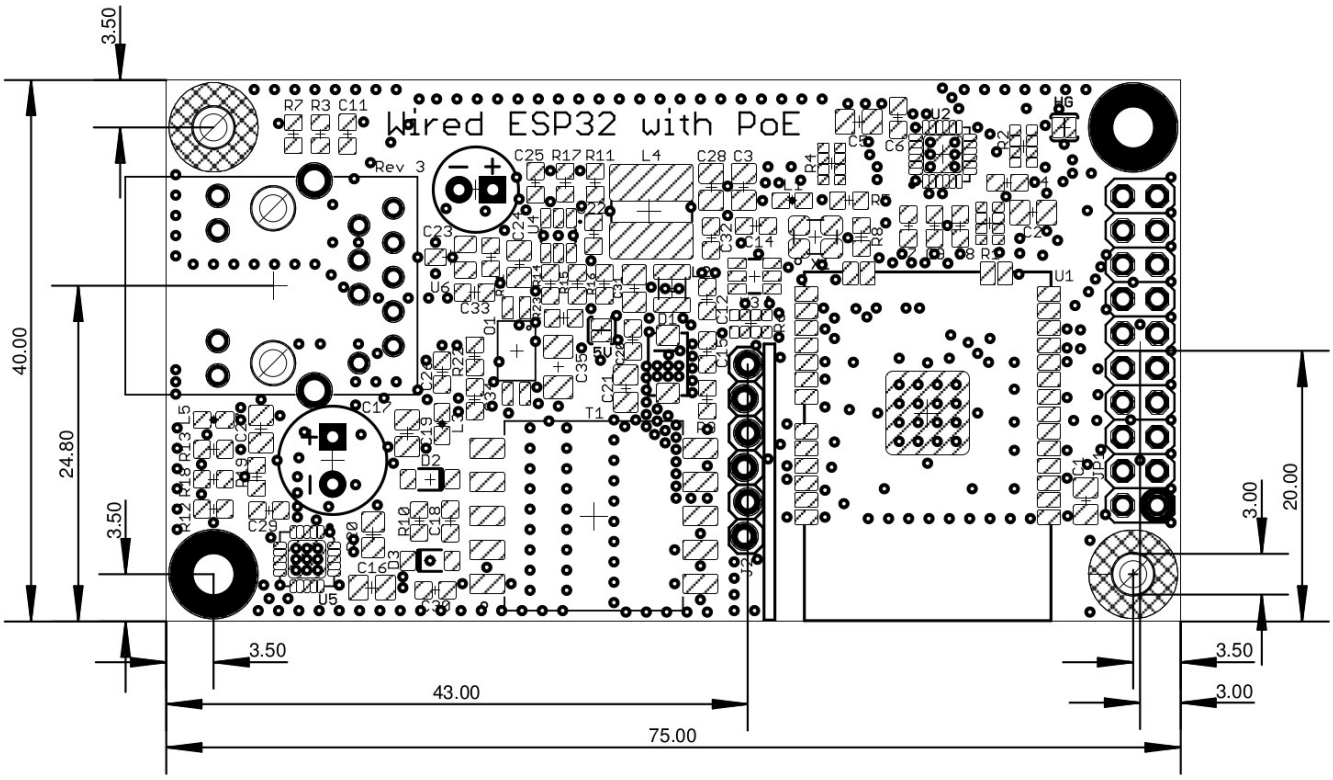
The board revision can be found next to the Ethernet jack. From revision 7 onward, a 4-layer PCB with 1 oz copper is used, while older revisions use a 2-layer PCB with 2 oz copper.

From revision 7 onward, the ESP32 module is an ESP32-WROOM-32E with 16 MB of flash, while older revisions used an ESP32-WROOM-32 module with 4 MB of flash. No software change is required, the boards with 16 MB of flash will work fine with images created for modules with 4 MB of flash, but the software has to be changed if the user wants to take advantage of the extra flash.

From revision 7 onward, the board uses an RTL8201FI Ethernet PHY chip for increased reliability versus older revisions that used a LAN8720AI Ethernet PHY. A software change is necessary to support the new PHY. In most cases, the only thing that changes is the definition of the PHY declared in the code.

Mechanical dimensions

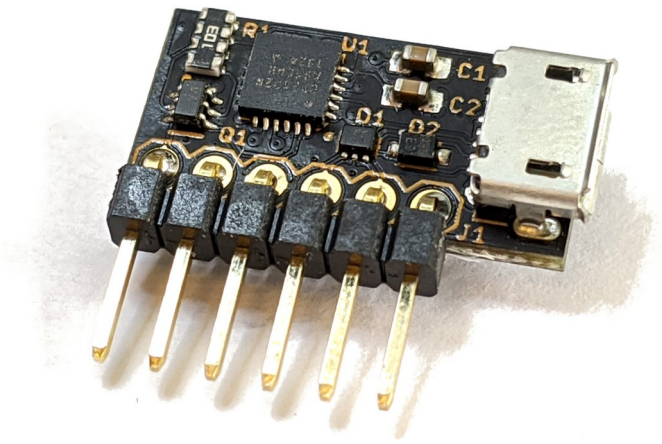
Below is a drawing with most of the relevant mechanical dimensions necessary for integration of the wESP32™.



The mounting holes are 3 mm in diameter, allowing the use of common M2.5 or 4-40 screws for mounting. The tallest component is the Ethernet jack, this is a Hanrun HR861153C or compatible jack. Including EMI tabs, this jack extends 15.2 mm above the PCB. The PCB is 1.2 mm in thickness and the jack's mechanical tabs and pins extend about 2 mm below the PCB. Keep in mind that some of these pins are electrically connected to the remote side of the Ethernet cable, and clearance needs to be maintained to any metal enclosure parts to maintain electrical isolation.

wESP32-Prog

A small programming / USB serial console submodule was created to be able to conveniently program the wESP32™ or to provide an optional USB serial port if required by the user's application. The tiny 22 mm x 15 mm module connects to the wESP32™ programming / serial console header footprint J2.



It features the Silicon Labs CP2102N chip for excellent cross platform driver support, on-board auto-program circuitry to transparently handle device programming and serial console use, a V+ blocking diode and ESD protection.

The user is free to use another programming device if they want, but this module is available for those who have no other ESP32 programming device or want a solution that was specifically designed to work well with the wESP32™.

Software support

Warning: GPIO pin configuration

By default, the wESP32™ ships with MicroPython and a `boot.py` script that automatically configures the GPIO pins correctly to communicate with the Ethernet PHY and enables the Ethernet subsystem. Plugging in an Ethernet cable should automatically connect the board to the network with DHCP.

If the user wants to use a different software environment, it is the user's responsibility to ensure that the software loaded on the ESP32 configures the GPIO pins that are connected to the Ethernet PHY correctly. Many of the signals between the ESP32 and the Ethernet PHY are part of the high speed RMII bus, and they are directly connected together without any protection. Pins involved in connectivity between the ESP32 and PHY are IO0, IO16, IO17, IO19, IO21, IO22, IO25, IO26, IO27.

If the user loads software on the ESP32 that does not configure the RMII pins correctly, it is possible the software will configure the pins in such a way that physical damage may result, for instance

connecting an output from the PHY to an output of the ESP32, resulting in a short circuit through the chip's output drivers. This may cause damage, at least to board revisions using the LAN8720. The new RTL8201 has never been damaged during our testing.

It is recommended the wESP32™ is first tested with the default MicroPython software to confirm network functionality, before other software is loaded. Basic network functionality can be confirmed by running `lan.ifconfig()` and checking that the wESP32™ received an IP address from DHCP. Defective units will not be replaced once they have been flashed with different software.

Espressif ESP-IDF

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

ESP-IDF is the official Espressif development framework for the ESP32. Since the ESP-IDF provides the ultimate flexibility, it is out-of-the-box compatible with the wESP32™. The only thing the user needs to do to make it work with the wESP32™ is set the right configuration for how the PHY is connected and configured (either in the `menuconfig` if using example projects, or in code):

- Ethernet PHY type is RTL8201 from revision 7 boards onward and LAN8720 for older revisions
- PHY interface is RMII
- PHY address is 0
- Clock mode is GPIO0 external input
- PHY power/reset pin is disabled (set to -1)
- SMI MDC pin is 16, SMI MDIO pin is 17

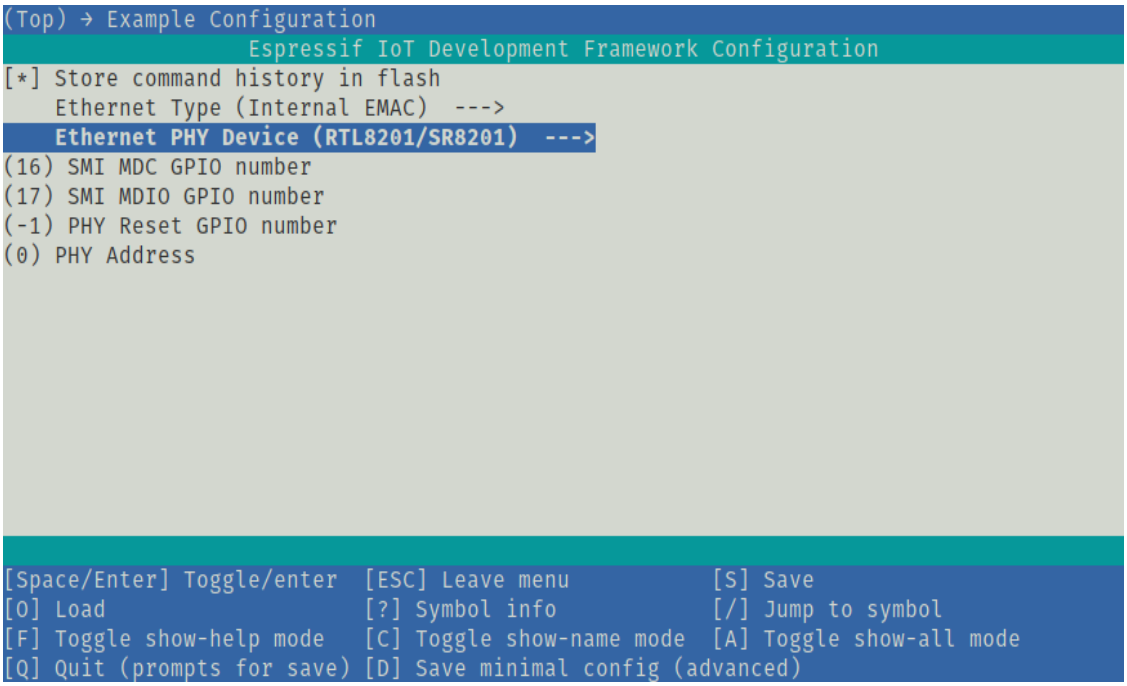
In code, this can be done with the following code snippet to configure and start Ethernet:

```
eth_mac_config_t mac_config = ETH_MAC_DEFAULT_CONFIG();
mac_config.smi_mdc_gpio_num = 16;
mac_config.smi_mdio_gpio_num = 17;
esp_eth_mac_t *mac = esp_eth_mac_new_esp32(&mac_config);

eth_phy_config_t phy_config = ETH_PHY_DEFAULT_CONFIG();
phy_config.phy_addr = 0;
phy_config.reset_gpio_num = -1;
// For boards revision 7 and onward, use RTL8201
esp_eth_phy_t *phy = esp_eth_phy_new_rtl8201(&phy_config);
// For boards before revision 7, use LAN8720
// esp_eth_phy_t *phy = esp_eth_phy_new_lan8720(&phy_config);
```

```
esp_eth_handle_t eth_handle = NULL;
esp_eth_config_t config = ETH_DEFAULT_CONFIG(mac, phy);
ESP_ERROR_CHECK(esp_eth_driver_install(&config, &eth_handle));

ESP_ERROR_CHECK(esp_netif_attach(eth_netif, esp_eth_new_netif_glue(eth_handle)));
ESP_ERROR_CHECK(esp_eth_start(eth_handle));
```

Full details can be found in the ESP-IDF Programming Guide (https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_eth.html).

For the ESP-IDF 'ethernet/iperf' and other examples, the correct configuration can be applied using `make menuconfig` under "Example Configuration":



Arduino-ESP32

https://github.com/espressif/arduino-esp32

The Arduino-ESP32 project allows you to program the wESP32™ from the popular Arduino IDE. Installation instructions are in the repo's `README.md`, support for wESP32™ boards is fully integrated into the stable release so you can use the Adruino Boards Manager installation instructions. Support for the RTL8201 used in revision 7 boards and onward requires version 2.0.0 or higher.

Once the ESP32 support is installed, you can select the wESP32™ from the Arduino IDE menu:

Tools → Board → Silicognition wESP32

To use the Ethernet connection in your sketch, you need to `#include <ETH.h>` and call `ETH.begin()` to start the Ethernet peripheral. This function can take optional parameters to set the configuration, the defaults are currently set for the wESP32™ before revision 7, for revision 7 you need to add parameters to select the RTL8201 PHY. It is likely that in the future, the default will be changed to work for the RTL8201 and extra parameters have to be specified for older LAN8720 boards.

Below is a minimal web server with mDNS example sketch:

```
#include <ETH.h>
#include <WebServer.h>
#include <ESPmDNS.h>

// Web server
WebServer server(80);

// HTTP handlers
void handleRoot() {
  server.send(200, "text/plain", "Hello from wESP32!\n");
}

void handleNotFound() {
  server.send(404, "text/plain", String("No ") + server.uri() + " here!\n");
}

void setup(){
  // Start the Ethernet, revision 7+ uses RTL8201
  ETH.begin(0, -1, 16, 17, ETH_PHY_RTL8201);
  // The defaults work for older revision boards
  // ETH.begin();
  // You can browse to wesp32demo.local with this
  MDNS.begin("wesp32demo");

  // Bind HTTP handler
  server.on("/", handleRoot);
  server.onNotFound(handleNotFound);

  // Start the Ethernet web server
  server.begin();
  // Add service to MDNS-SD
  MDNS.addService("http", "tcp", 80);
}

void loop(){
  server.handleClient();
}
```

MicroPython

https://github.com/micropython/micropython/

The MicroPython project provides a rich Python 3 environment for small microcontrollers. The ESP32 and Ethernet are well supported, and due to its flexible nature, the wESP32™ is supported out-of-the-box.

The following is the minimum code needed to get a wESP32™ before revision 7 that used the LAN8720 PHY connected to Ethernet with MicroPython:

```
import machine
import network

lan = network.LAN(mdc = machine.Pin(16), mdio = machine.Pin(17), power = None,
phy_type = network.PHY_LAN8720, phy_addr = 0)
lan.active(1)
```

This works with ESP-IDF v3.x based MicroPython releases and with ESP-IDF v4.x based MicroPython releases from version v1.16 onward.

For revision 7 and above wESP32™ boards, MicroPython v1.16 or newer based on ESP-IDF v4.x is required to support the RTL8201. The following is the minimum code needed when using this PHY:

```
import machine
import network

lan = network.LAN(mdc = machine.Pin(16), mdio = machine.Pin(17), power = None,
phy_type = network.PHY_RTL8201, phy_addr = 0)
lan.active(1)
```

From MicroPython version 1.17 on, there is a custom build of MicroPython for the wESP32™ revision 7 to make use of the full 16 MB of flash available on the board. The “Generic ESP32 module” binaries will still work, but only provide access to 4 MB of flash. The MicroPython website does not yet have a download page specific to the wESP32™. To get the wESP32™ build, copy the URL of a “Generic ESP32” binary and add a “w” at the start of the filename. For instance, the download link of version 1.17 for the wESP32™ is:

https://micropython.org/resources/firmware/**w**esp32-20210902-v1.17.bin

Once connected, all networking functionality will occur over Ethernet.

The wESP32™ by default ships with MicroPython and a pre-installed `boot.py` script that contains the code shown above, configured for the correct PHY populated on a particular board revision. This

means that out-of-the-box, the wESP32™ will connect to the network when the Ethernet jack is connected without any code needing to be supplied by the user.

Lua-RTOS-ESP32

https://github.com/whitecatboard/Lua-RTOS-ESP32

Lua-RTOS-ESP32 provides a rich Lua environment for the ESP32. Installation instructions are in the `README.md`, when prompted to enter a board type choose the option for the "Silicognition wESP32" with or without OTA. As of the time of writing, this project is built on IDF 3.x and only supports board revisions before revision 7 that use the LAN8720 Ethernet PHY.

Once built and flashed, the minimum code to get connected over Ethernet is:

```
net.en.setup()
net.en.start()
```

Once connected, all networking functionality will occur over Ethernet. Since this runtime has dropbear on board, you can even remotely log in to the Lua REPL over SSH!

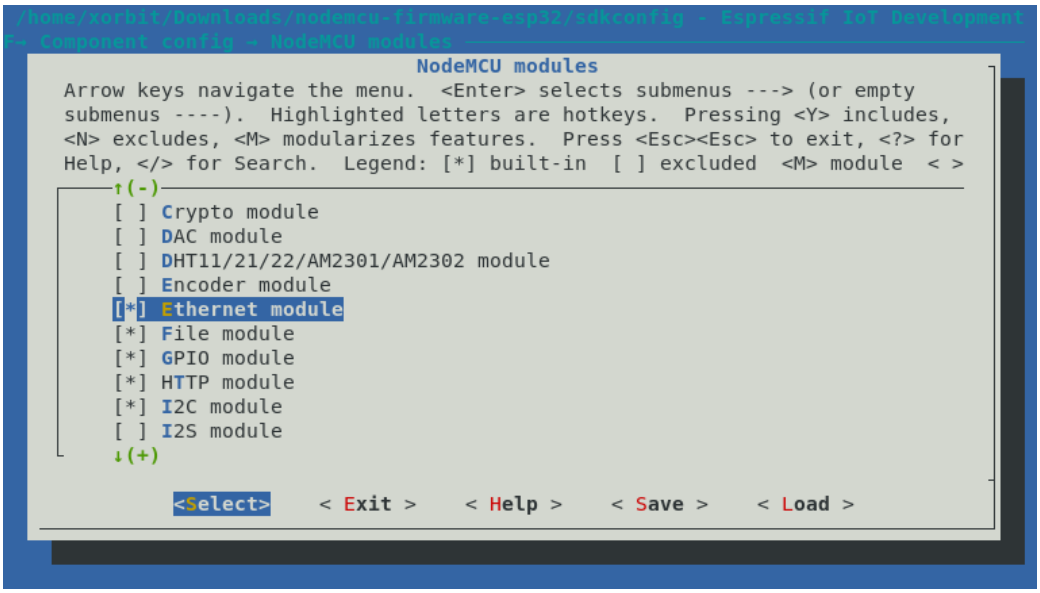
NodeMCU

https://github.com/nodemcu/nodemcu-firmware/tree/dev-esp32-idf4

Fans of Lua have a second option to run Lua on the wESP32™, and this one works with revision 7 and newer. The `dev-esp32-idf4` branch has to be used as it supports both the LAN8720 on older revisions and the RTL8201 on revision 7. They provide quite a nice implementation with events that allow the user to respond to various states of the Ethernet connection such as cable connect / disconnect and acquiring an IP address.

NodeMCU does not seem to provide ready-made images, you have to check out the `dev-esp32-idf4` branch from Github and follow the build instructions found in the project's documentation. To be able to use the Ethernet port, you have to ensure you turn on the Ethernet module when you run `make menuconfig` to configure your build options:

Component config → NodeMCU modules → Ethernet module



Once built and flashed, the minimum code to get connected over Ethernet when using a board before revision 7 is:

```
eth.init({phy = eth.PHY_LAN8720, addr = 0, mdc = 16, mdio = 17})
```

For revision 7 and later boards using the RTL8201, instead use:

```
eth.init({phy = eth.PHY_RTL8201, addr = 0, mdc = 16, mdio = 17})
```

Mongoose OS

https://mongoose-os.com/

Mongoose OS is a development framework for IoT firmware, with good support for the ESP32. It comes with a lot of back end functionality to take care of device management, such as a device dashboard and OTA updates.

Ethernet on the ESP32 is supported, but at the time of writing, the project does not seem to tie it in to the network stack in any way to make it useful. See this issue on Github.

Hoping that this is a temporary problem, below is an example `mos.yml` file that contains the right bits to configure a project to make use of the wESP32™ hardware:

```
author: mongoose-os
description: A JS-enabled demo Mongoose OS firmware
platform: esp32
```

```
version: 1.0
manifest_version: 2017-05-18
libs_version: ${mos.version}
modules_version: ${mos.version}
mongoose_os_version: ${mos.version}

config_schema:
  - ["mqtt.server", "iot.eclipse.org:1883"]
  - ["i2c.enable", true]

tags:
  - js

filesystem:
  - fs

config_schema:
  - ["eth.clk_mode", 0]
  - ["eth.mdc_gpio", 16]
  - ["eth.mdio_gpio", 17]
  - ["eth.phy_addr", 0]
  - ["eth.enable", true]

libs:
  - location: https://github.com/mongoose-os-libs/ethernet
  - location: https://github.com/mongoose-os-libs/js-demo-bundle
```

Note in particular the `config_schema` section that configures the Ethernet subsystem and the `ethernet` library in the `libs` section. This is sufficient for wESP32™ revisions prior to 7 using the LAN8720, from revision 7 onward, the following part needs to be added to configure the Ethernet driver to use the RTL8201 PHY:

```
conds:
  - when: mos.platform == "esp32"
    apply:
      cdefs:
        MGOS_ETH_PHY_LAN87x0: 0
        MGOS_ETH_PHY_RTL8201: 1
```

Compatibility

The wESP32™ was created to work with standard compliant IEEE 802.3af and 802.3at PoE power sourcing equipment (PSE). It is not compatible with so-called "passive PoE" systems on the market but requires a minimum voltage of 37V and power negotiation as per the IEEE specification. Category 3 or higher rated cable is required.

While cost may be somewhat higher for standard compliant equipment, it is preferable because a lot of though was put in to the standard by industry experts to ensure that it *works well and is safe*, not just in bench or lab settings, but also in actual production installations. Transmitting power over long wire runs is just not compatible with low 12 V or 24 V line voltages and devices lacking isolation, and the fact that these non-compliant hackish solutions exist unfortunately results in customer confusion when "PoE" devices don't work together.

It should be noted that since the wESP32™ is powered from a third party device over a potentially long cable, across possibly a large number of connecting points, performance in reaching the specified 12.95 W of power is not solely dependent on the wESP32™ itself but also on many external factors.

Electrical characteristics

Unless otherwise indicated, all characteristics apply for V_{IN} = 37 V – 57V and T_A = 0 ºC to 50 ºC. Typical values are at 25 ºC and V_{IN} = 48 V.

Parameter	Sym	Min	Typ	Max	Unit	Conditions
PoE input voltage	V_{IN}	37	48	57	V	
V+ output voltage	V_{+}		12			"5V" solder jumper open
V+ output voltage (5V option)	V_{+OPT}		5			"5V" solder jumper bridged
V+ external input voltage	V_{+EXT}			15	V	V+ supplied from JP1
Total output power	P_{OUT}		12.95		W	
V+ output power	P_{OUT_V+}		12.5		W	"5V" solder jumper open
V+ output power (5V option)	P_{OUT_V+OPT}		5		W	"5V" solder jumper closed
Logic supply voltage	V_{DD}		3.3		V	
Logic supply power	P_{VDD}		6		W	"5V" solder jumper open, V+ supplied from PoE

Disclaimer

We have taken every precaution to provide a high performance, safe product but do not accept any liability or responsibility for your use of it. It is your responsibility as a customer to use the device in a proper and sensible way, and show proper respect for the high amount of energy present, and maintaining isolation. By using the device you agree that Silicognition LLC will not be held liable for any damages you may incur due to your use of the device.

Sales and support

To buy the wESP32™, please visit http://wesp32.com. To order in quantity and for volume discounts, please contact sales@wesp32.com.

For technical support, please contact support@wesp32.com.

X-ON Electronics

Largest Supplier of Electrical and Electronic Components

Click to view similar products for WiFi Development Tools - 802.11 ***category:***

Click to view products by Crowd Supply ***manufacturer:***

Other Similar products are found below :

YSAEWIFI-1 SKY65981-11EK1 QPF7221PCK-01 SIMSA915C-Cloud-DKL SIMSA433C-Cloud-DKL ISM43903-R48-EVB-E QPF4206BEVB01 RN-G2SDK SKY85734-11EK1 SKY85735-11EK1 MIKROE-2336 EVAL_PAN1760EMK 3210 ATWINC1500-XPRO 2471 DM990001 WRL-13711 2999 ATWILC3000-SHLD DFR0321 TEL0118 3213 DFR0489 SLWSTK-COEXBP WRL-13804 DEV-13907 UP-3GHAT-A20-0001 3405 TEL0078 2680 2702 2821 3606 3653 3654 4178 4285 CS-ANAVI-25 CS-ANAVI-26 CS-ANAVI-23 CS-ANAVI-24 CS-ANAVI-28 CS-ANAVI-29 CS-ANAVI-30 CS-ANAVI-31 ABX00021 ABX00023 AKX00004 AKX00014 AKX00015