Espruino on ESP32

Note: *This page documents running the Espruino firmware on the ESP32 board.

Warning: Espruino on the ESP32 defaults to 115200 baud on its serial interface. This means you will need to adjust this setting in the IDE if you use that. (Other Espruino ports default to 9600 baud.)

Contents

Overview

The ESP32 is a dual core Tensilica LX6 microcontroller with 520 KB SRAM, integrated Wifi, Bluetooth, and more. Espruino is a very lightweight JavaScript interpreter that runs on the ESP32, and other microcontrollers.

This documentation is intended for those who want to run JavaScript on any ESP32 microcontrollers. It will describe how to flash the ESP32 with the latest firmware, connect to Wifi and get the other pins going.

There are a few different boards that contain the ESP32 microcontroller. These instructions below should be generic enough for all boards, but you may have to adapt the instructions in places.

Quick links

- Download the latest ESP32 firmware release (From v1.92 onward)
- Download 'cutting edge' ESP32 firmware - these may not always work
- Espruino ESP32 Forum - Main support forum
- ESP32 Forum
- Developer chat for Espruino ESP32 - Focuses on development issues.

ESP32 Features

- 240 MHz dual core Tensilica LX6 microcontroller
- Built-in Wifi and Bluetooth (classic and BLE)
- 2.2V to 3.6V operating voltage
- 32 GPIO pins:
  - 3x UARTs, including hardware flow control

    - 3x SPI
    - 2x I2S
    - 12x ADC input channels
    - 2x DAC
    - 2x I2C
    - PWM/timer input/output available on every GPIO pin
    - Supports external SPI flash up to 16 MB
    - SD-card interface support
- Sensors: Ultra-low noise analog amplifier, Hall sensor, 10x capacitive touch interface, 32 kHz crystal oscillator

Limitations / Work in Progress

Supported by Espruino on the ESP32:

- onewire
- hardware SPI
- hardware I2C
- DAC
- ADC
- Serial
- WIFI - as a client and access point
- Bluetooth LE - BLE

Not supported by Espruino on the ESP32 (yet):

- No Over-The-Air (OTA) firmware updates.
- Bluetooth Classic

Known Issues

- Espruino Web IDE had issues with the ESP32 back in 2017, see below.

Getting Started

Installing drivers

Depending on the board and operating system you have, you might want to install the FTDI or Silicon Labs USB driver:

- FTDI
- Silicon Labs CP210x

Flashing

If your device is not already flashed, below are the instructions. Flashing involves downloading the latest firmware to your PC and then copying via USB cable to the microcontroller.

Note: This flashing process is being improved. We expect to remove the build stage and provide the firmware as a download.

Building Firmware and installing the toolchain

You can get the Espruino firmare from the Travis cutting-edge builds. Get the file ending in `_esp32.bin`.

You can also build the Espruino firmware yourself:

The following work in a bash shell environment, you will need git, and other essential build tools (e.g. on Ubuntu run `sudo apt-get install build-essential`).

```
# Get the Espruino source code
git clone https://github.com/espruino/Espruino.git
cd Espruino
# Download and set up the toolchain ('source' is important here)
source scripts/provision.sh ESP32
# Clean and rebuild
make clean && BOARD=ESP32 make
```

You will have a file called `espruino_esp32.bin`. This is your ESP32 firmware.

The actual flash

Download esptool if you haven't downloaded it already.

To flash the ESP32 we use `esptool.py`. Before you run `esptool.py`, make sure you know the flashing procedure for the board you have. Some boards have a flash button, that you press when you are running the flash utility. For boards such as the Dev Kit C board - The DTS and RTS lines are used and put the board into bootloader mode automatically

PyCom boards

for PyCom boards with the ESP32 microcontroller you need to:

1. connect the ESP32 board to your PC using the USB cable.
2. hold pin G23 to GND;
3. run the following command; and then
4. press the reset button.

ESP-WROOM-32 (plain, no dev board)

1. connect pins as follows

```
VCC -> VCC(3.3V)
GND -> GND
EN -> VCC
TX -> RX (on usb controller)
RX -> TX (on usb controller)
IO0 -> GND (only for flash)
```

2. run the following command; and then
3. leave IO0 floating (not connected) and reset chip

ESP-WROOM-32 (dev board)

Flashing can be done by USB, no special actions on pins are needed.

Flashing command

The following command will write the flash to the ESP32. Note you need to select the correct port, on Windows it will be something like `COM3`.

Initial flash

```
esp-idf/components/esptool_py/esptool/esptool.py    \
        --chip esp32                                \
        --port /dev/ttyUSB0                         \
        --baud 921600                               \
        --after hard_reset write_flash              \
        -z                                          \
        --flash_mode dio                            \
        --flash_freq 40m                            \
        --flash_size detect                         \
        0x1000 bootloader.bin                       \
        0x8000 partitions_espruino.bin              \
        0x10000 espruino_esp32.bin
```

If the `bootloader.bin` and `partitions_espruino.bin` files are not included from the same source you acquired the `espruino_esp32.bin` build, they should be available in the corresponding `espruino_1v92.*_esp32.tgz` package of the Travis cutting-edge builds.

Subsequent updates

This example is to replace existing Espruino firmware after a new release:

```
esp-idf/components/esptool_py/esptool/esptool.py    \
        --chip esp32                                \
        --port /dev/ttyUSB0                         \
        --baud 921600                               \
```

```
        --before esp32r0                        \
        --after hard_reset write_flash          \
        -z                                      \
        --flash_mode dio                        \
        --flash_freq 40m                        \
        --flash_size detect                     \
        0x10000 espruino_esp32.bin
```

You should see something like the following:

```
esptool.py v2.0-beta1
Connecting.....................
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 921600
Changed.
Attaching SPI flash...
Configuring flash size...
Auto-detected Flash size: 4MB
Compressed 894288 bytes to 517230...
Wrote 894288 bytes (517230 compressed) at 0x00010000 in 8.4 seconds (effective 848.5 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting...
```

Your device is flashed.

Espruino Web IDE

The Espruino Web IDE is a basic development environment that allows you to write your code and deploy it to the ESP32.

Issue: In 2017, there was an issue with Espruino Web IDE, sometimes it will not connect. Especially the first time you try. The workaround is to use another tool to connect to the ESP32, like minicom or cutecom, see below. Once connected using one of these tools, try again using Espruino Web IDE.

Once you have connected once and enabled wifi on boot (see below), you will be able to connect to the ESP32 via telnet, using Espruino Web IDE, this tends to be quite reliable.

If these instructions were still relevant for you in 2022 or later, please edit the docs to reflect that it's an ongoing issue.

minicom / CuteCom

There are other methods to connect to your ESP32, not just Espruino Web IDE. Two very basic tools are minicom, CuteCom, and screen which may already be installed on your system. Another cross platform terminal program with a friendly interface is CoolTerm.

screen is usually used for multiplexing terminals, keeping terminal sessions alive while you're logged out, and so forth. However, it will also function as a serial terminal

```
screen /dev/ttyUSB0 115200
```

Minicom is a basic console based tool that allows you to connect to a serial device, such as the ESP32. Below is the command to get you connected to the ESP32.

```
minicom --baudrate 115200 --device /dev/ttyUSB0
```

CuteCom is also a basic GUI tool that allows you to connect to a serial device. Run CuteCom and connect via the correct port.

Running some basic JavaScript

Once you have the espruino console running you will be able run JavaScript directly in the console. You should see a > character with a flashing cursor, this indicates that espruino is waiting for input. If you don't see it press the enter key.

Type the following:

```
console.log('Hello ESP32!');
```

It will output the following:

```
Hello ESP32!
=undefined
```

You should have expected the "`Hello ESP32!`" text, but probably didn't expect "`=undefined`". This is normal and it indicates the result of the last operation, which in this case is the return value of `console.log`, which is always `undefined`.

Connecting to Wifi

If your board with the ESP32 has an antenna connection point, connect your Wifi antenna. Many boards have an in-built antenna so this may not be required.

```
var ssid = 'YOUR_SSID';
var password = 'YOUR_SSID_PASSWORD';

var wifi = require('Wifi');
wifi.connect(ssid, {password: password}, function() {
    console.log('Connected to Wifi.  IP address is:', wifi.getIP().ip);
    wifi.save(); // Next reboot will auto-connect
});
```

Additional commands:

- `wifi.save()`: This will save the Wifi details such that the ESP32 will reconnect to the wifi after a reboot.
- `wifi.scan((ap) => { console.log(ap) })`: List all the access points seen by the ESP32.
- `wifi.setHostname('myESP32')`: set the hostname. (Not currently implemented.)

Note: The ESP32's Wifi implementation supports both a simple access point mode and a station mode. The station mode is highly recommended for normal operation as the access point mode is very limited. It supports 4 stations max and offers no routing between stations.

Note: You need a good 3.3v regulator with a solid power supply. If you get errors as soon as Wifi starts it's probably because the power is insufficient. A 500-600mA regulator with at least 22uF capacitor is recommended.

Creating a basic web-server

Once you have wifi going you will be able to create a simple web server.

```
var ssid = 'YOUR_SSID';
var password = 'YOUR_SSID_PASSWORD';
var port = 80;

function processRequest (req, res) {
  res.writeHead(200);
  res.end('Hello World');
}

wifi.connect(ssid, {password: password}, function() {

    var http = require('http');
    http.createServer(processRequest).listen(port);

    console.log(`Web server running at http://${wifi.getIP().ip}:${port}`)
});
```

This will output something like `Web server running at http://10.42.0.119` and you will be able to browse to that address.

Bluetooth

See https://www.espruino.com/BLE+Communications

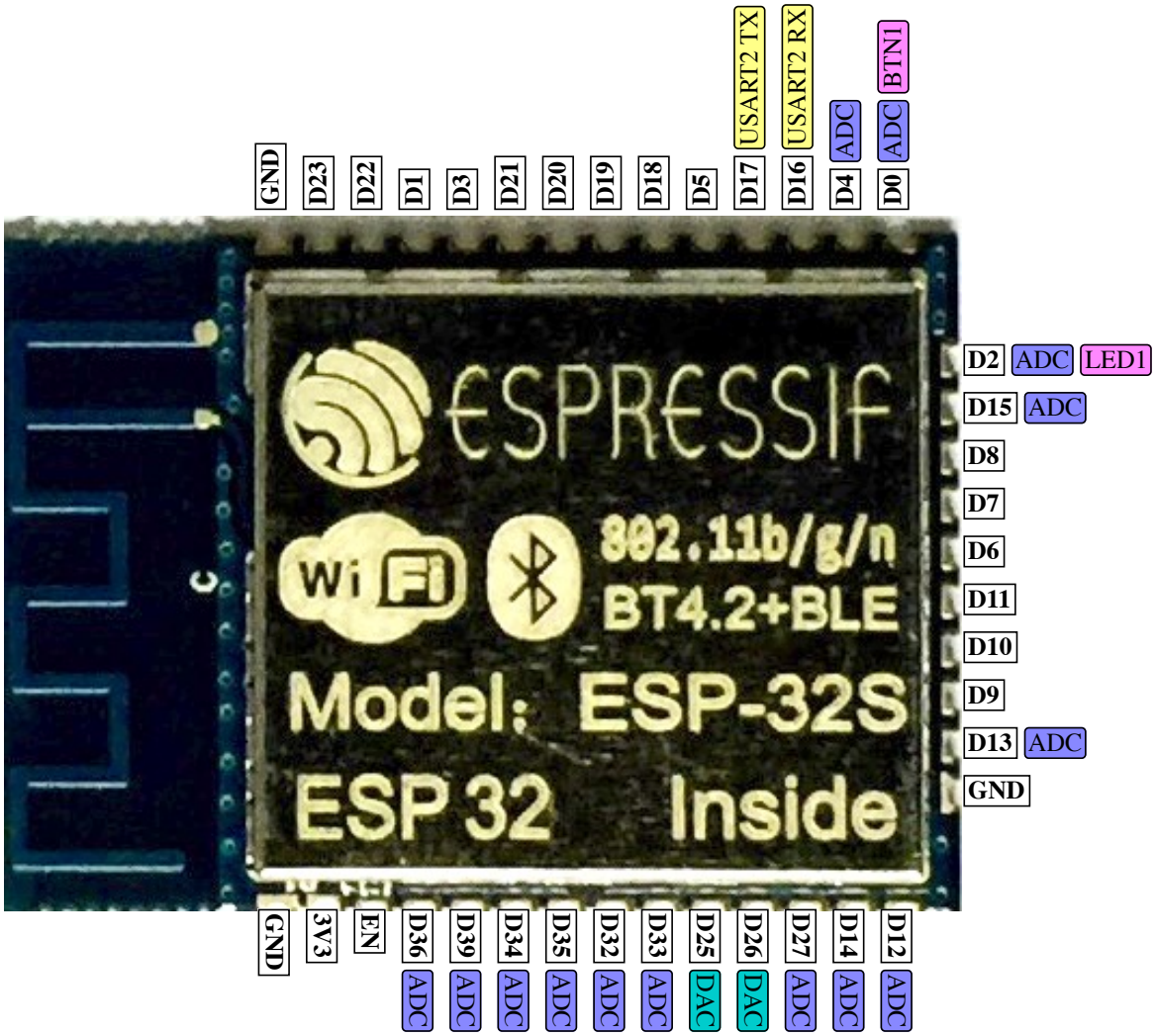
GPIO Pins

The ESP32 has 40 GPIO pins. These are available in Espruino as the variables D0 to D40, each of these variables are instances of the Pin class.

Hover the mouse over a pin function for more information. Clicking in a function will tell you how to use it in Espruino.

- Purple boxes show pins that are used for other functionality on the board. You should avoid using these unless you know that the marked device is not used.
- ! boxes contain extra information about the pin. Hover your mouse over them to see it.
- 3.3v boxes mark pins that are not 5v tolerant (they only take inputs from 0 - 3.3v, not 0 - 5v).
- GND is ground (0v).
- ADC is an Analog to Digital Converter (for reading analog voltages)

- DAC is a Digital to Analog Converter (for creating analog voltages). This is not available on all boards.
- USART is a 2 wire peripheral for Serial Data.



Pins not on connectors

D24
D28
D29
D30
D31
D37 ADC
D38 ADC

For example, connect GPIO25 and D14 with a wire. Then run the following code.

```
D14.mode('output');
D25.mode('input');
console.log('Before: ', D25.read());
D14.set();                  // Set the D14 pin to output a HIGH value.
console.log('After: ', D25.read());
```

This will output:

```
Before:  false
After:  true
```

Note: The ESP32 GPIO pins support totem-pole and open-drain outputs, and they support a weak internal pull-up resistor (in the 20KOhm-50KOhm range). The Espruino D0 through D32 pins map directory to GPIO0 through GPIO32 on the ESP32. Remember that GPIO6 through GPIO11 are used for the external flash chip and are therefore not available. Also, GPIO0 and GPIO2 must be pulled-up at boot and GPIO15 must be pulled-down at boot.

The ESP32 ADC function is available on any pin (D0-D15) but really uses a separate pin on the ESP32 (this should be changed to an A0 pin).

I2C Example

Below is an example of the HD44780 LCD controller. It uses the I2C class.

```
I2C1.setup({ 'scl': D17,
             'sda': D16,
            bitrate: 100000 });
var lcd = require('HD44780').connectI2C(I2C1);
lcd.print('Hello ESP32!!');
```

SPI Example

Below is an example for the PCD8544 LCD controller (48 × 84 pixels matrix LCD). It uses the SPI class.

```
/*
Display     ESP-32   DEF Colour
LED         3V3      N/A
SCLCK       GPIO4    Green    IO05
DN<MOSI>    GPIO16   Yellow
D/C         GPIO17   Orange
SCE         GPIO5    Red
RST         GPIO18   Brown
GND         GND      Grey
VCC         3V3      Blue
BackLight   GND      Purple
*/

SPI2.setup({ sck:D5, mosi:D23 });

var g = require('PCD8544').connect(SPI1,
    D17 /* RS / DC */,
    D18 /* CS / CE */,
    D16 /*RST*/, function() {
  g.clear();
  g.setRotation(2); //Flip display 180
  g.drawString('Hi Esp32',0,0);
  g.drawLine(0, 10, 84, 10);
  g.flip();
});
```

Serial / UART Example

There are two serial ports (UARTs) available for use on the ESP32 with Espruino. These are accessed throught the Serial class. The **`Serial1`** and **`Serial2`** functions are available for use. Note that if you are accessing the ESP32 via serial terminal **`Serial1`** may already be used, hence it's not available (use Telnet if this is an issue).

For the example below, connect a wire from the `GPIO04` (transmit pin, TX) and `GPIO15` (receive pin, RX).

```
Serial2.setup(9600, { tx: D4, rx: D15 });
Serial2.on('data', function(data) {
    console.log('Serial2: ', data);
});
Serial2.print('Hello UART');
```

This will output:

```
Serial2: Hello UART
```

Note: The ESP32 has two UARTS. UART0 (**`Serial1`**) uses `GPIO10` for TX and `GPIO32` for RX and is used by the Espruino JavaScript console. It can be used for other things once the Espruino console is moved to another device. For instance calling **`LoopbackA`**`.setConsole()` will move the console to 'loopback' (where is can be accessed using **`LoopbackB`**), and will free up **`Serial1`** for use like any normal Espruino Serial port.

Digital Read / Write Example

PWM on most pins. True DAC on GPIO25 and GPIO26.

First version of PWM uses ledc driver. Usually, this matches needs for frequency much more than sigmadelta. Mapping of PWM channel to timer is done in a special way. There are 5 channel usable, as long as 5Khz are ok for frequency. Channels are internally controlled. If you try to use more than 5, you will get a message. Left 3 channels are designed to support a frequency of your choice, as long as it is between 1hz and 78 Khz. If you try to use a 4th channel with a frequency, other than 5Khz you will a meesage.

Problem

Handling of PWM with those 5/3 method is not always easy to follow. As long as we don't have a better solution, its better to have this than nothing. Limitation of frequency is boring, but as long as we use it for analog output, it should be acceptable. Please have in mind, PWM via analog always need a lowpass filter, usually a simple RC.

Analog Read / Write Example

There are two ADC channels but only ADC-1 is support at the moment. The ADC-1 channel is accessible through `IO35` and `IO36(VP_SENSOR)`. ADC PINs should not be connected to a voltage higher than VCC - if a higher input voltage is needed, a resistor should be added between the input and the ADC and then the real voltage could be calculated by including the resistor's voltage drop in the equation.

usage example: reading analog voltage from IO35

```
var reading = analogRead(D35);
// reading is a float between 0 and 1
var estimatedInputVoltage = reading * 3.3;
```

more information on espruino ADC docs and analogRead method

Unique Identifier for the ESP32

The ESP32 does not have a serial number. It does have two mac addresses "burned-in", which one can use for identification purposes. `getSerial()` returns the MAC address of the STA interface.

System time

From a JavaScript perspective, we can get and set the system time using the JS functions called `getTime()` and `setTime()`. These get and take a time in seconds (float).

digitalPulse implementation

There are 8 channel available. Each one with is connected to a pin. Timimg goes down to micro secs. Implemented with the esp-idf RMT driver.

setWatch implementation

These pins can be watched:

- D0
- D12 .. D19
- D21, D22
- D25 .. D27
- D34 .. D39

Saving code to flash

Currently 64KB of flash are reserved to save JS code to flash using the save() function. This area is also used for `E.saveBootCode()`

If the save() area contains something that crashes Espruino or otherwise doesn't let you reset the system you can disable whatever is saved by flashing blank.bin to 0x100000.

Flash map and access

Note: if you are looking for a free flash area to use, call `require( "Flash").getFree()`, which will return a list of available areas (see docs).

Current ESP32 modules are 4Mb.

The result of all this is the following:

Start	Length	Function
0x000000	4KB	Secure Boot
0x008000	4KB	Bootloader
0x009000	16KB	nvs - esp-idf non-volatile storage area
0x00d000	8KB	otadata - esp-idf keeps track on which the current firmware to boot
0x010000	960KB	factory - the initial espruino firmware.
0x100000	64KB	js_code - the saved espruino interpreter and `E.setBootCode()`
0x200000	1024KB	ota_0 - area for ota updates (not yet implemented)
0x300000	1024KB	1Mb Flash Fat Filesystem

This is defined in partitions_espruino.csv in the EspruinoBuildTools repository

The last 1Mb has been defined as a flash FAT filesystem. It needs to be intialised for first use - this does it in a safe way that won't delete existing files:

```
try {
  fs.readdirSync();
 } catch (e) { //'Uncaught Error: Unable to mount media : NO_FILESYSTEM'
  console.log('Formatting FS - only need to do once');
  E.flashFatFS({ format: true });
}
```

you can then use the filesystem with fs or File objects:

```
fs.writeFileSync('hello world.txt', 'This is the way the world ends\nHello World\nnot with a bang but a whimper.\n');
fs.readFileSync('hello world.txt');
```

Further reading

TODO: Add additional resources not already covered.

Troubleshooting

Guru Meditation Error

Symptom:

```
Guru Meditation Error of type InstrFetchProhibited occurred on core  0. Exception was unhandled.
Register dump:
PC      : 0xffffffff  PS      : 0x00060c30  A0      : 0x8008e036  A1      : 0x3ffd54b0
A2      : 0x3f400148  A3      : 0x3ffb6952  A4      : 0x00000008  A5      : 0xffffffe8
A6      : 0xfffffffa4 A7      : 0x3ffb1fa0  A8      : 0xffffffff  A9      : 0x3ffd54c0
A10     : 0x00000000  A11     : 0x00000002  A12     : 0x5fff0007  A13     : 0x00000000
A14     : 0x00000000  A15     : 0x00000000  SAR     : 0x00000000  EXCCAUSE: 0x00000014
EXCVADDR: 0xfffffffc  LBEG    : 0x4000c2e0  LEND    : 0x4000c2f6  LCOUNT  : 0x00000000
```

Cause: if it happens whenever wifi tries to connect then the chip is not getting enough power. **Solution:** use a designated power source (not the USB serial adapter), add capacitors to the power line

Official Espruino Boards

Bangle.js 2

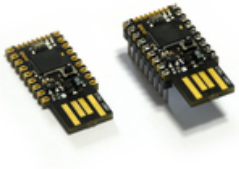
Puck.js

Espruino WiFi

Pixl.js Multicolour

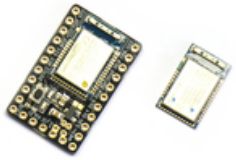
Pixl.js

Espruino Pico



Original Espruino Board

MDBT42Q Bluetooth Module



Bangle.js

This page is auto-generated from GitHub. If you see any mistakes or have suggestions, please let us know.