ESP32-JCI-R

User Manual

Prerelease version 0.1
Espressif Systems

About This Guide

This document is intended to help users set up the basic software development environment for developing applications using hardware based on the ESP32-JCI-R module.

Release Notes

Date	Version	Release notes
2020.7	V0.1	Preliminary release.

Documentation Change Notification

Espressif provides email notifications to keep customers updated on changes to technical documentation. Please subscribe at www.espressif.com/en/subscribe.

Certification

Download certificates for Espressif products from www.espressif.com/en/certificates.

Table of Contents

8. Introduction to ESP32-JCI-R

8.1. ESP32-JCI-R

ESP32-JCI-R is a powerful, generic Wi-Fi+BT+BLE MCU module that targets a wide variety of applications, ranging from low-power sensor networks to the most demanding tasks, such as voice encoding, music streaming and MP3 decoding.

At the core of this module is the ESP32-D0WD-V3 chip. The chip embedded is designed to be scalable and adaptive. There are two CPU cores that can be individually controlled, and the CPU clock frequency is adjustable from 80 MHz to 240 MHz. The user may also power off the CPU and make use of the low-power co-processor to constantly monitor the peripherals for changes or crossing of thresholds. ESP32 integrates a rich set of peripherals, ranging from capacitive touch sensors, Hall sensors, SD card interface, Ethernet, high-speed SPI,UART, I^{2}S and I^{2}C.

The integration of Bluetooth, Bluetooth LE and Wi-Fi ensures that a wide range of applications can be targeted, and that the module is future proof: using Wi-Fi allows a large physical range and direct connection to the internet through a Wi-Fi router, while using Bluetooth allows the user to conveniently connect to the phone or broadcast low energy beacons for its detection. The sleep current of the ESP32 chip is less than 5 µA, making it suitable for battery powered and wearable electronics applications. ESP32 supports a data rate of up to 150 Mbps, and 20 dBm output power at the antenna to ensure the widest physical range. As such the chip does offer industry-leading specifications and the best performance for electronic integration, range, power consumption, and connectivity.

The operating system chosen for ESP32 is freeRTOS with LwIP; TLS 1.2 with hardware acceleration is built in as well. Secure (encrypted) over the air (OTA) upgrade is also supported, so that developers can continually upgrade their products even after their release.

8.2. ESP-IDF

The Espressif IoT Development Framework (ESP-IDF for short) is a framework for developing applications based on the Espressif ESP32. Users can develop applications in Windows/Linux/MacOS based on ESP-IDF.

8.3. Preparation

To develop applications for ESP32-JCI-R you need:

- PC loaded with either Windows, Linux or Mac operating system
- Toolchain to build the Application for ESP32
- ESP-IDF that essentially contains API for ESP32 and scripts to operate the toolchain
- A text editor to write programs (Projects) in C, e.g., Eclipse
- The ESP32 board itself and a USB cable to connect it to the PC

9. Get Started with ESP32-JCI-R

9.1. Toolchain Setup

The quickest way to start development with ESP32 is by installing a prebuilt toolchain. Pick up your OS below and follow provided instructions.

- Windows
- Linux
- Mac OS

📖 ***Note:***

*We are using **~/esp** directory to install the prebuilt toolchain, ESP-IDF and sample applications. You can use different directory, but need to adjust respective commands.*

Depending on your experience and preferences, instead of using a prebuilt toolchain, you may want to customize your environment. To set up the system your own way go to section Customized Setup of Toolchain.

Once you are done with setting up the toolchain then go to section Get ESP-IDF.

9.2. Get ESP-IDF

Besides the toolchain (that contains programs to compile and build the application), you also need ESP32 specific API / libraries. They are provided by Espressif in ESP-IDF repository.

To get it, open terminal, navigate to the directory you want to put ESP-IDF, and clone it using `git clone` command:

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

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

📖 ***Note:***

Do not miss the `--recursive` *option. If you have already cloned ESP-IDF without this option, run another command to get all the submodules:*

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

9.3. Set up Path to ESP-IDF

The toolchain programs access ESP-IDF using ***IDF_PATH*** environment variable. This variable should be set up on your PC, otherwise projects will not build. Setting may be done manually, each time PC is restarted. Another option is to set up it permanently by defining ***IDF_PATH*** in user profile. To do so, follow instructions in Add IDF_PATH to User Profile.

10. Start a Project

Now you are ready to prepare your application for ESP32. To start off quickly, we will use hello_world project from ***examples*** directory in IDF.

Copy ***get-started/hello_world*** to ***~/esp*** directory:

```
cd ~/esp
cp -r $IDF_PATH/examples/get-started/hello_world .
```

You can also find a range of example projects under the ***examples*** directory in ESP-IDF. These example project directories can be copied in the same way as presented above, to begin your own projects.

📖 ***Note:***

The ESP-IDF build system does not support spaces in paths to ESP-IDF or to projects.

11. Connect

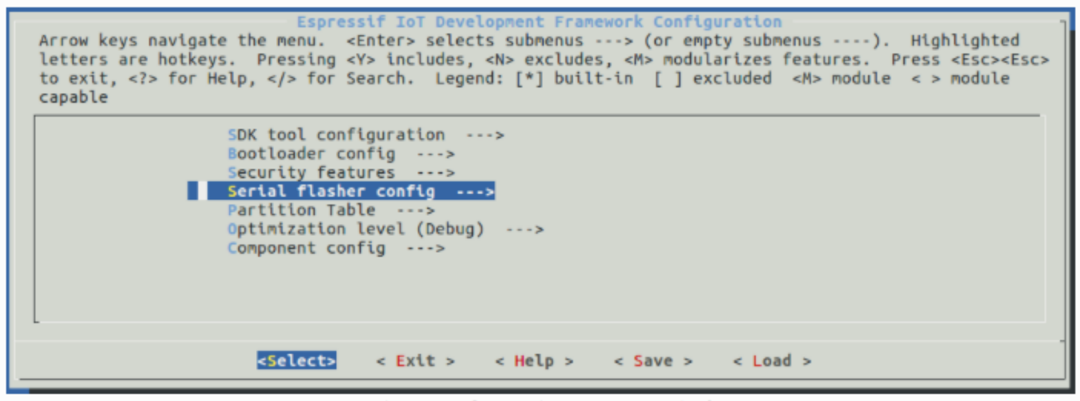
You are almost there. To be able to proceed further, connect ESP32 board to PC, check under what serial port the board is visible and verify if serial communication works. If you are not sure how to do it, check instructions in Establish Serial Connection with ESP32. Note the port number, as it will be required in the next step.

12. Configure

Being in terminal window, go to directory of ***hello_world*** application by typing `cd ~/esp/hello_world`. Then start project configuration utility ***menuconfig***:

```
cd ~/esp/hello_world
make menuconfig
```

If previous steps have been done correctly, the following menu will be displayed:



```
Espressif IoT Development Framework Configuration
Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted
letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc>
to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module
capable

                         SDK tool configuration  --->
                         Bootloader config  --->
                         Security features  --->
                         Serial flasher config  --->
                         Partition Table  --->
                         Optimization level (Debug)  --->
                         Component config  --->

              <Select>    < Exit >    < Help >    < Save >    < Load >
```

Project configuration - Home window

In the menu, navigate to ***Serial flasher config*** > ***Default serial port*** to configure the serial port, where project will be loaded to. Confirm selection by pressing enter, save configuration by selecting ***<Save>*** and then exit application by selecting ***<Exit>***.

📖 ***Note:***

On Windows, serial ports have names like COM1. On MacOS, they start with /dev/cu.. On Linux, they start with /dev/tty. (See Establish Serial Connection with ESP32 for full details.)

Here are couple of tips on navigation and use of ***menuconfig***:

- Use ***up*** & ***down*** arrow keys to navigate the menu.
- Use ***Enter*** key to go into a submenu, ***Escape*** key to go out or to exit.
- Type ***?*** to see a help screen. Enter key exits the help screen.
- Use Space key, or ***Y*** and ***N*** keys to enable (Yes) and disable (No) configuration items with checkboxes "***[*]***".
- Pressing ***?*** while highlighting a configuration item displays help about that item.
- Type ***/*** to search the configuration items.

📖 ***Note:***

If you are Arch Linux user, navigate to ***SDK tool configuration*** *and change the name of* ***Python 2 interpreter*** *from* ***python*** *to* ***python2***.

13. Build and Flash

Now you can build and flash the application. Run:

```
make flash
```

This will compile the application and all the ESP-IDF components, generate bootloader, partition table, and application binaries, and flash these binaries to your ESP32 board.

```
esptool.py v2.0-beta2

Flashing binaries to serial port /dev/ttyUSB0 (app at offset 0x10000)...

esptool.py v2.0-beta2

Connecting........___

Uploading stub...

Running stub...

Stub running...

Changing baud rate to 921600

Changed.

Attaching SPI flash...

Configuring flash size...

Auto-detected Flash size: 4MB

Flash params set to 0x0220

Compressed 11616 bytes to 6695...

Wrote 11616 bytes (6695 compressed) at 0x00001000 in 0.1 seconds (effective 920.5 kbit/
s)...

Hash of data verified.

Compressed 408096 bytes to 171625...

Wrote 408096 bytes (171625 compressed) at 0x00010000 in 3.9 seconds (effective 847.3
kbit/s)...

Hash of data verified.

Compressed 3072 bytes to 82...

Wrote 3072 bytes (82 compressed) at 0x00008000 in 0.0 seconds (effective 8297.4 kbit/
s)...

Hash of data verified.


Leaving...

Hard resetting...
```

If there are no issues, at the end of build process, you should see messages describing progress of loading process. Finally, the end module will be reset and “hello_world” application will start.

If you'd like to use the Eclipse IDE instead of running make, check out Build and Flash with Eclipse IDE.

14. Monitor

To see if "hello_world" application is indeed running, type `make monitor`. This command is launching [IDF Monitor]() application:

```
$ make monitor
MONITOR
--- idf_monitor on /dev/ttyUSB0 115200 ---
--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
ets Jun  8 2016 00:22:57
...
```

Several lines below, after start up and diagnostic log, you should see "Hello world!" printed out by the application.

```
...
Hello world!
Restarting in 10 seconds...
I (211) cpu_start: Starting scheduler on APP CPU.
Restarting in 9 seconds...
Restarting in 8 seconds...
Restarting in 7 seconds...
```

To exit the monitor use shortcut ***Ctrl+]***.

📖 ***Note:***
If instead of the messages above, you see a random garbage, or monitor fails shortly after upload, your board is likely using 26MHz crystal, while the ESP-IDF assumes default of 40MHz. Exit the monitor, go back to the menuconfig, change CONFIG_ESP32_XTAL_FREQ_SEL to 26MHz, then build and flash the application again. This is found under make menuconfig under Component config –> ESP32-specific –> Main XTAL frequency.

To execute `make flash` and `make monitor` in one go, type `make flash monitor`. Check section [IDF Monitor]() for handy shortcuts and more details on using this application.

That's all what you need to get started with ESP32!

Now you are ready to try some other examples, or go right to developing your own applications.