JTAG Debugging for ESP32



Version 1.0 Copyright © 2017

About This Document

This document provides a guide to installing OpenOCD for ESP32 and debugging using gdb in a Linux virtual machine. The document is structured as follows:

Chapter	Title	Content
Chapter 1	Introduction	Introduction to the purpose of this document.
Chapter 2	JTAG Adapter Hardware	Introduction to JTAG adapters for ESP32-enabled variant of OpenOCD.
Chapter 3	Environment Setup for JTAG	Introduction to environment setup for JTAG debugging.
Chapter 4	Debugging Example	A guide to debugging using gdb with the example of esp-idf/ examples/02_blink .
Chapter 5	Notes	Notes on OpenOCD's support for ESD-IDF FreeRTOS and ESP32 quirks.

Release Note

Date	Version	Release notes
2017.03	V1.0	Initial release.

Table of Contents

1.	Intro	troduction1					
2.	JTAC	Adapter Hardware	2				
3.	Envi	ronment Setup for JTAG	3				
		Install ESP-IDF					
	3.2.	Install OpenOCD	3				
	3.3.	Set up Environment for gdb					
1	. Debugging Example7						
4.	Deni	Debugging Example					
5.	Note	9S	.15				
	5.1.	FreeRTOS Support	15				
		ESP32 Quirks					



1.

Introduction

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

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

Espressif has ported OpenOCD to support the ESP32 processor and the multicore FreeRTOS, which will be the foundation of most ESP32 apps, and has written some tools to help with features OpenOCD does not support natively.

This document provides a guide to installing OpenOCD for ESP32 and debugging using gdb in a Linux virtual machine.

Note:

- Please note that this document is meant as a workaround for lack of a Windows version of OpenOCD as
 yet. Users with access to a Linux or Mac system are recommended to OpenOCD directly without
 resorting to virtual machines. For updated information on JTAG debugging for ESP32, please refer to:
 https://github.com/espressif/esp-idf/blob/master/docs/openocd.rst.
- This document is written based on tests using ESP-WROVER-KIT.



2.

JTAG Adapter Hardware

Users will need a JTAG adapter that is compatible with both the voltage levels on the ESP32 and with the OpenOCD software. The JTAG port on the ESP32 is an industry-standard JTAG port which lacks (and does not need) the TRST pin. All JTAG I/O pins are powered from the VDD_3P3_RTC pin (which normally would be powered by a 3.3V rail), so the JTAG adapter needs to be able to work with JTAG pins in that voltage range.

On the software side, OpenOCD supports a fair amount of JTAG adapters. Please see http://openocd.org/doc/html/Debug-Adapter-Hardware.html for an (unfortunately slightly incomplete) list of the adapters OpenOCD works with. This page lists SWD-compatible adapters as well; note that the ESP32 does not support SWD.

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

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



3. Environment Setup for JTAG

3.1. Install ESP-IDF

Please install Git in your Linux virtual machine first.

sudo apt-get install git

Note:

- The prerequisite for successful environment setup is that the virtual machine can access extranet.
- Please refer to <u>ESP-IDF Getting Started Guide</u> for compiling a sample code before installing OpenOCD. In Chapter 4, esp-idf\examples\02_blink is used as the sample code for ESP32 JTAG debugging guide.

3.2. Install OpenOCD

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

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

cd openocd-esp32

git submodule init

git submodule update
```

2. Execute the following command in the command terminal on Linux.

```
sudo apt-get install make

sudo apt-get install libtool

sudo apt-get install pkg-config

sudo apt-get install autoconf

sudo apt-get install automake

sudo apt-get install texinfo

sudo apt-get install libusb-1.0
```

Note:

- Version of pkg-config should be 0.2.3 or above.
- Version of autoconf should be 2.6.4 or above.
- Version of automake should be 1.9 or above.
- When using USB-Blaster, ASIX Presto, OpenJTAG and FT2232 as adapters, drivers libFTDI and FTD2XX need to be downloaded and installed.
- When using CMSIS-DAP, HIDAPI is needed.
- 3. Input the following commands on the Linux virtual machine.

./bootstrap



./configure

make

Notice:

- Should errors occur, please try again until the command make works. If there is a submodule problem from OpenOCD, please cd to the directory openocd-esp3 and input git submodule update --init.
- If the commands are successfully run, information of the enabled JTAG will be printed. If the information of your device is not shown in the log, please use ./configure to enable your device with reference to ../openocd-esp32/doc/INSTALL.txt.
- For details concerning compiling OpenOCD, please refer to openocd-esp32/README.

3.3. Set up Environment for gdb

- Connect JTAG to ESP32 and to the PC. Wait for the installation of the driver. When the
 driver is successfully installed, select your device on the virtual machine and click on it to
 connect it to the virtual machine.
- 2. Open another command prompt window on the virtual machine. Connect Minicom to the serial port and wait for printing.
 - Install Minicom;
 - Input the command sudo apt-get install minicom;
 - Configure Minicom with the command sudo minicom -s and you will see the following prompt:

```
+----[configuration]----+
| Filenames and paths
| File transfer protocols
| Serial port setup
| Modem and dialing
| Screen and keyboard
| Save setup as dfl
| Save setup as..
| Exit
| Exit from Minicom
```

- Select "Serial port setup" and the following prompt will show up:



- Reconfigure Serial Device, connects Minicom to the device UART, and disable the hardware flow control; save the configuration as default and as DFL.
- Input the command minicom. If Minicom is correctly configured, output log on the terminal will show that the ESP32 device is running. (Please make sure there are programs running in the flash of ESP32.)

3. Invoke OpenOCD:

- Copy esp32.cfg in esp-idf/docs to the openocd-esp32 directory;
- Edit the copied *esp32.cfg* file. Most importantly, change the *source* [find interface/ftdi/tumpa.cfg] line to reflect the physical JTAG adapter connected. (Note that this step is not needed when using the ESP-WROVER-KIT.)
- Open a terminal and cd to the openocd-esp32 directory;
- Run ./src/openocd -s ./tcl -f ./esp32.cfg to start OpenOCD.
- If OpenOCD is successfully invoked, you should see the following output:

user@machine:~/esp32/openocd-esp32\$./src/openocd -s ./tcl/ -f ../openocd-esp32-tools/esp32.cfg

Open On-Chip Debugger 0.10.0-dev-00446-g6e13a97-dirty (2016-08-23-16:36)

```
Licensed under GNU GPL v2

For bug reports, read

http://openocd.org/doc/doxygen/bugs.html

none separate

adapter speed: 200 kHz

Info: clock speed 200 kHz

Info: JTAG tap: esp32.cpu0 tap/device found: 0x120034e5 (mfg: 0x272 (Tensilica), part: 0x2003, ver: 0x1)

Info: JTAG tap: esp32.cpu1 tap/device found: 0x120034e5 (mfg: 0x272 (Tensilica), part: 0x2003, ver: 0x1)

Info: esp32.cpu0: Debug controller was reset (pwrstat=0x5F, after clear 0x0F).

Info: esp32.cpu0: Core was reset (pwrstat=0x5F, after clear 0x0F).
```



- If you see an error indicating permission problems, please see the "Permissions delegation" part in the *OpenOCD README*.
- If you see JTAG errors (...all ones/...all zeroes), please check your connections and see if everything is powered on.
- 4. Invoke gdb.
 - Run the following command:

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

- If the command works, the following printed log will be shown:

The environment for JTAG is now set up on the Linux virtual machine.

Note:

If you have compiled the ESP32 toolchain using Crosstool-NG, or if you have downloaded a precompiled toolchain from the Espressif website, you should already have xtensa-esp32-elf-gdb, a version of gdb that can be used for this.

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



4.

Debugging Example

This chapter provides a guide to debugging using gdb with the example of *esp-idf/examples/02_blink*.

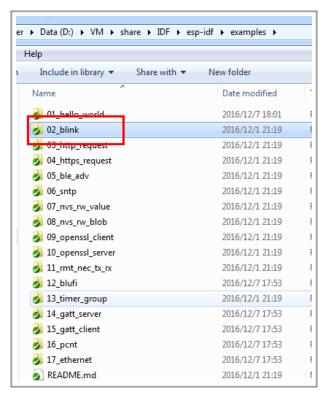


Figure 4-1. esp-idf/examples/02_blink

 Input minicom on terminal. The following output log will show. (Print is added in the program of 02_blink.)

```
Welcome to minicom 2.7

OPTIONS: I18n
Compiled on Feb 7 2016, 13:37:27.
Port /dev/ttyUSB1, 13:12:04

Press CTRL-A Z for help on special keys

8
9
10
11
12
13
```



2. Run the OpenOCD command ./src/openocd -s ./tcl -f ./esp32.cfg. The following printed log will show up:

```
🛿 🖨 📵 chushuchen@chushuchen-VirtualBox: /mnt/Share/openocd-esp32
t sent! (1553). Workaround: increase "set remotetimeout" in GDB
chushuchen@chushuchen-VirtualBox:/mnt/Share/openocd-esp32$ ./src/openocd -s ./tc
l -f ./esp32.cfg
Open On-Chip Debugger 0.10.0-dev-g3098897 (2016-11-18-11:49)
Licensed under GNU GPL v2
For bug reports, read
          http://openocd.org/doc/doxygen/bugs.html
none separate
adapter speed: 200 kHz
force hard breakpoints
Error: no device found
Error: unable to open ftdi device with vid 0403, pid 8a98, description '*', seri al '*' at bus location '*'
Info: clock speed 200 kHz
Info : JTAG tap: esp32.cpu0 tap/device found: 0x120034e5 (mfg: 0x272 (Tensilica)
, part: 0x2003, ver: 0x1)
Info : JTAG tap: esp32.cpu1 tap/device found: 0x120034e5 (mfg: 0x272 (Tensilica)
, part: 0x2003, ver: 0x1)
Info : esp32.cpu0: Debug controller was reset (pwrstat=0x5F, after clear 0x0F).
Info : esp32.cpu0: Core was reset (pwrstat=0x5F, after clear 0x0F).
Info : esp32.cpu1: Debug controller was reset (pwrstat=0x5F, after clear 0x0F).
Info : esp32.cpu1: Core was reset (pwrstat=0x5F, after clear 0x0F).
```

3. If the printing continues, please try pressing the reset button on the ESP32 board.

```
t sent! (1553). Workaround: Increase "set renotetineout" in GOB

chushuchen@chushuchen-VirtualBox:/mnt/Share/openocd-esp322

t sent! (1553). Workaround: Increase "set renotetineout" in GOB

chushuchen@chushuchen-VirtualBox:/mnt/Share/openocd-esp325./src/openocd -s./tc

l - /esp32.cfg

open on-Chip Debugger 0.10.0-dev-g3098897 (2016-11-18-11:49)

tleensed under CNU CPL v2

for buy reports, read

none separate

adapter speed: 200 bitz

force hard breakpoints

fror: unable to open fidd device with vid 0403, pid 04098, description '*', seri

al '''' Info: clock speed 200 kitz

info: 1156 taps: (sp32.cpub tap/device found: 0x120034e5 (nfg: 0x272 (Tensilica)

part: 0x2003, ver: 0x1)

info: 116 taps: (sp32.cpub tap/device found: 0x120034e5 (nfg: 0x272 (Tensilica)

part: 0x2003, ver: 0x1)

info: esp32.cpub tap/device found: 0x120034e5 (nfg: 0x272 (Tensilica)

part: 0x2003, ver: 0x1)

info: esp32.cpub tap/device found: 0x120034e5 (nfg: 0x272 (Tensilica)

part: 0x2003, ver: 0x1)

info: esp32.cpub tap/device found: 0x120034e5 (nfg: 0x272 (Tensilica)

part: 0x2003, ver: 0x1)

info: esp32.cpub tap/device found: 0x120034e5 (nfg: 0x272 (Tensilica)

part: 0x2003, ver: 0x1)

info: esp32.cpub tap/device found: 0x120034e5 (nfg: 0x272 (Tensilica)

part: 0x2003, ver: 0x1)

info: esp32.cpub tabus seest (purstate0x5F, after clear 0x0F).

info: esp32.cpub: 0x003 vex seest (purstate0x5F, after clear 0x0F).

info: esp32.cpub: 0x003 vex seest (purstate0x5F, after clear 0x0F).
```

When the program is halted in the OpenOCD window, as the screenshot below shows, and the terminal stops printing, it means that the OpenOCD is invoked.

4. Run the command below in the 02 blink window.

```
xtensa-esp32-elf-gdb -ex 'target remote localhost:3333' ./build/blink.elf
```



The following log will show up:

```
Detaching from program: /mnt/Share/02_blink/build/blink.elf, Remote target Remote communication error. Target disconnected.: Connection reset by peer. chushuchen@chushuchen-VirtualBox:/mnt/Share/02_blink$ xtensa-esp32-elf-gdb -ex 'target remote localhost:3333' ./build/blink.elf
GNU gdb (crosstool-NG 8d95cad) 7.10
Copyright (C) 2015 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <a href="http://gnu.org/licenses/gpl.html">http://gnu.org/licenses/gpl.html</a>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-build_pc-linux-gnu --target=xtensa-esp 32-elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<a href="http://www.gnu.org/software/gdb/bugs/">http://www.gnu.org/software/gdb/bugs/</a>,
Find the GDB manual and other documentation resources online at:
<a href="http://www.gnu.org/software/gdb/documentation/">http://www.gnu.org/software/gdb/documentation/</a>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./build/blink.elf...done.
Remote debugging using localhost:3333
ipc_task (arg=0x0) at /mnt/Share/IDF/esp-idf/components/esp32/./ipc.c:45
45
{
(gdb) }
{
```

As the screenshot above shows, the program is halted at line 45 of ipc_task.

5. Set a breakpoint to halt the program before blink_task.

Input the following command to set breakpoints in gdb window.

```
(gdb) break blink_task
```

The following log will output:

```
Breakpoint 1 at 0×400d8888: file/mnt/Share/02_blink/main/./blink.c, line 23.
```

It means that breakpoint 1 is set in the 23rd line in the function blink.c.

6. Run the command continue or c in gdb window, and the program will stop at breakpoint 1.

```
(gdb) c
Continuing.
Cannot execute this command without a live selected thread.
```

If the information in the above screenshot shows, choose one thread to run.

• Use the gdb command i threads to check information about threads:



```
(gdb) i threads
[New Thread 1073448124]
       Target Id
                           Frame
       Thread 1073448124 (No Name) ipc_task (arg=0x0)
    at /mnt/Share/IDF/esp-idf/components/esp32/./ipc.c:45
       Thread 1073436808 (No Name : Running) ipc_task (arg=0x0)
    at /mnt/Share/IDF/esp-idf/components/esp32/./ipc.c:45
       Thread 1073438616 (No Name) ipc_task (arg=0x0)
    at /mnt/Share/IDF/esp-idf/components/esp32/./ipc.c:45
       Thread 1073444456 (No Name) main_task (args=0x0)
    at /mnt/Share/IDF/esp-idf/components/esp32/./cpu_start.c:234
Thread 1073450092 (Svc) prvTimerTask (pvParameters=0x0)
    at /mnt/Share/IDF/esp-idf/components/freertos/./timers.c:427
       Thread 1073446404 (No Name) prvIdleTask (pvParameters=0x0)
    at /mnt/Share/IDF/esp-idf/components/freertos/./tasks.c:3224
(gdb)
```

• Input a gdb command to choose the thread to run. For example, input thread 3 to run thread 3, as the following screenshot shows:

```
(gdb) thread 3
[Switching to thread 3 (Thread 1073450092)]
#0 prvTimerTask (pvParameters=0x0)
    at /mnt/Share/IDF/esp-idf/components/freertos/./timers.c:427
427 {
```

• Input command c to continue running the program:

```
(gdb) c
Continuing.
esp32.cpu0: Target halted, pc=0x400D8888
[New Thread 1073450980]
[Switching to Thread 1073450980]

Breakpoint 1, blink_task (pvParameter=0x0)
    at /mnt/Share/02_blink/main/./blink.c:23
23     {
(gdb) ■
```

As the screenshot above shows, the program stops at breakpoint 1, in the 23rd line of *blink.c*.

7. Input command list or 1 to see the code.



```
(gdb) list

18 or you can edit the following line and set a number here.

19 */
20 #define BLINK_GPIO CONFIG_BLINK_GPIO

21

22 void blink_task(void *pvParameter)

23 {

24  /* Configure the IOMUX register for pad BLINK_GPIO (some pads are
25  muxed to GPIO on reset already, but some default to other
26  functions and need to be switched to GPIO. Consult the
27  Technical Reference for a list of pads and their default

(gdb)
```

Continue inputing command 1 to see the rest of the code.

```
Technical Reference for a list of pads and their default
27
(gdb) l
28
                   functions.)
29
               gpio_pad_select_gpio(BLINK_GPIO);
30
               /* Set the GPIO as a push/pull output */
31
               gpio_set_direction(BLINK_GPIO, GPIO_MODE_OUTPUT);
32
                    \overline{int} = 0;
33
34
               while(1) {
35
                    /* Blink off (output low) */
36
                    gpio_set_level(BLINK_GPIO, 0);
37
                    vTaskDelay(1000 / portTICK_RATE_MS);
(gdb) l
                    printf("%d\n",i++);
/* Blink on (output high) */
gpio_set_level(BLINK_GPIO, 1);
vTaskDelay(1000 / portTICK_RATE_MS);
38
39
40
41
42
43
          }
44
45
          void app_main()
46
47
               nvs_flash_init();
(gdb)
```

8. To set a new breakpoint in line 40, input the command break 40. Run the command continue or c, and the program will stops at line 40.

9. Input print i or p i to check the value of variable i in the codes.

```
(gdb) p i
$1 = 1
```



10. Run the command c and the program will stop at breakpoint 2 while looping. The printed log of the serial port will show the value of variable i, which is the same as the value returned by the command p i.

```
rst:0x1 (POWERON_RESET),boot:0x16 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0x00
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3ffc0008,len:0
load:0x3ffc0008,len:1964
load:0x40078000,len:3668
load:0x40080000,len:260
I (338) heap_alloc_caps: Initializing heap allocator:
I (339) heap_alloc_caps: Region 19: 3FFB5008 len 0002AFF8 tag 0
I (340) heap_alloc_caps: Region 25: 3FFE8000 len 00018000 tag 1
I (350) cpu_start: Pro cpu up.
I (356) cpu_start: Starting app cpu, entry point is 0x40080ae4
   (0) cpu_start: App cpu up.
(371) cpu_start: Pro cpu start user code
(395) rtc: rtc v160 Nov 22 2016 19:00:05
(406) rtc: XTAL 40M
   (811) phy: phy_version: 258, Nov 29 2016, 15:51:07, 0, 0 (1272) cpu_start: Starting scheduler on PRO CPU.
   (158760) cpu_start: Starting scheduler on APP CPU.
     chushuchen@chushuchen-VirtualBox: /mnt/Share/02_blink
uction generated an exception!
Exception reading fcr!
esp32.cpu0: Target halted, pc=0x400D88A9
esp32.cpu0: Target halted, pc=0x400D88A6
Breakpoint 2, blink_task (pvParameter=<optimized out>)
at /mnt/Share/02_blink/main/./blink.c:37
vTaskDelay(1000 / portTICK_RATE_MS);
 (gdb) p i
$2 = 1
(gdb) c
Continuing.
keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent!
(5216). Workaround: increase "set remotetimeout" in GDB esp32.cpu0: Target halted, pc=0x400D88A9
esp32.cpu0: Target halted, pc=0x400D88A6
Breakpoint 2, blink_task (pvParameter=<optimized out>)
at /mnt/Share/02_blink/main/./blink.c:37

vTaskDelay(1000 / portTICK_RATE_MS);
(gdb) p i
$3 = 2
 (gdb)
```

Note:

Please restart OpenOCD and gdb when having problems executing the program.

11. Use the command n to step execution, or command s to step into a function.



```
esp32.cpu0: Target halted, pc=0x400D88BA esp32.cpu0: Target halted, pc=0x400D88BC
esp32.cpu0: Target halted, pc=0x400D8918
Warning:
Cannot insert breakpoint 0.
Cannot access memory at address 0x400d88bf
gpio set level (gpio num=GPIO NUM 0, level=1)
    at /mnt/Share/IDF/esp-idf/components/driver/./gpio.c:161
161
(gdb) s
esp32.cpu0: esp108_fetch_all_regs (line 649): DSR (800D88BF) indicates target st
ill busy!
esp32.cpu0: esp108_fetch_all_regs (line 649): DSR (800D88BF) indicates DIR instr
uction generated an exception!
esp32.cpu0: esp108_fetch_all_regs (line 649): DSR (800D88BF) indicates DIR instr
uction generated an overrun!
Exception reading pc!
esp32.cpu0: esp108_fetch_all_regs (line 673): DSR (400D891B) indicates DIR instr
uction generated an exception!
esp32.cpu0: esp108_fetch_all_regs (line 673): DSR (400D891B) indicates DIR instr
uction generated an overrun!
Exception reading pc!
esp32.cpu0: esp108_fetch_all_regs (line 673): DSR (8000CC13) indicates DIR instr
uction generated an exception!
Exception reading fcr!
keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent!
(5005). Workaround: increase "set remotetimeout" in GDB
esp32.cpu0: Target halted, pc=0x400D891B
162
            GPIO_CHECK(GPIO_IS_VALID_GPIO(gpio_num), "GPIO number error", ESP_ER
R INVALID ARG);
(gdb)
```

- 12. Since no more than two breakpoints can be set in an ESP32 device, please delete breakpoints already set in order to set new breakpoints.
 - Use the command info break to check the breakpoints that have been set.

```
(gdb) info break
Num
                       Disp Enb Address
                                            What
        Type
                                0x400d8888 in blink_task
        breakpoint
                       keep y
                                            at /mnt/Share/02 blink/main/./blink.c
:23
        breakpoint already hit 1 time
2
                       keep y 0x400d88b8 in blink_task
        breakpoint
                                            at /mnt/Share/02_blink/main/./blink.c
:40
        breakpoint already hit 2 times
(gdb)
```

• Use the command delete 1 to delete breakpoint 1.

```
(gdb) delete 1
(gdb) info break
Num Type Disp Enb Address What
2 breakpoint keep y 0x400d88b8 in blink_task
at /mnt/Share/02_blink/main/./blink.c
:40
breakpoint already hit 2 times
(gdb) ■
```



Table 4-1. Commands to Delete Breakpoints

Command	Description	Example
delete	Function: delete [breakpoints num] [range] delete: deletes a breakpoint or a set of breakpoints.	delete 5 delete 1-10
clear	Function: • clear deletes multiple breakpoints in the current line. • clear location clear: deletes all breakpoints at a designated location. location: the location of the breakpoints to delete.	clear list_insert //Delete all the breakpoints in the function list_insert. clear list.c:list_delet //Delete all the breakpoints in the function list_delet in list.c. clear 12 //Delete all breakpoints in line 12. clear list.c:12 //Delete all breakpoints in line 12 in list.c.

Set new breakpoints.

Note:

The above-mentioned commands will meet basic debugging needs. For more gdb commands, please consult online resources.

Espressif 14/16 2017.03



5. Notes

5.1. FreeRTOS Support

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

5.2. ESP32 Quirks

- Normal gdb breakpoints (b myFunction) can only be set in IRAM, because that memory
 is writable. Setting these types of breakpoints in code will not work in flash. Instead, use
 a hardware breakpoint (hb myFunction). The ESP32 supports 2 hardware breakpoints. It
 also supports two watchpoints, so two variables can be watched for change or read by
 the gdb command watch myVariable.
- Because gdb only sees the system from the point of view of the selected CPU, only the FreeRTOS tasks that are suspended and the task running on the CPU, to which gdb is connected, will be shown correctly. The task that was active on the other CPU can be inspected, but its state may be wildly inconsistent.
- The ESP-IDF code can compile in various support options for OpenOCD: it can stop execution when the first thread is started, and break the system if a panic or unhandled exception is thrown. Both options are enabled by default but can be disabled using the *esp-idf* configuration menu. Please see the *make menuconfig* menu for more details.
- Normally, under OpenOCD, a board can be reset by entering mon reset or mon reset halt into gdb. For the ESP32, these commands work more or less, but they also have side effects. First of all, an OpenOCD reset only resets the CPU cores, not the peripherals, which may lead to undefined behaviour if software assumes the after-reset state of peripherals. Secondly, mon reset halt stops before FreeRTOS is initialized. OpenOCD assumes a running FreeRTOS (in the default configuration, you can change this by editing esp32.cfg), and may get confused.



Espressif IOT Team www.espressif.com

Disclaimer and Copyright Notice

Information in this document, including URL references, is subject to change without notice.

THIS DOCUMENT IS PROVIDED AS IS WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

All liability, including liability for infringement of any proprietary rights, relating to the use of information in this document, is disclaimed. No licenses express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

The Wi-Fi Alliance Member logo is a trademark of the Wi-Fi Alliance. The Bluetooth logo is a registered trademark of Bluetooth SIG.

All trade names, trademarks and registered trademarks mentioned in this document are property of their respective owners, and are hereby acknowledged.

Copyright © 2017 Espressif Inc. All rights reserved.