

06 — ESP32 (ESP-IDF) Integration

TraxProbe & Traxcope User Manual · Version 2.0.0 · Updated 2026-10-07

This chapter adds TraxProbe to an ESP-IDF project on an Espressif Xtensa chip and streams it to Traxcope. You get FreeRTOS kernel tracing on both cores, plus logs, variables, and the rest of the TraxProbe API.

ESP-IDF differs from the generic setup in Chapter 02 in four ways:

- TraxProbe is an **ESP-IDF component**. Its `CMakeLists.txt` is already in the SDK.
- The kernel trace hooks come in through ESP-IDF's **ESP Trace** external-library option. You do not edit `FreeRTOSConfig.h`.
- Metadata placement uses an ldgen fragment, `linker/esp/trax_meta.ld`, not `trax_probe.ld`. The component registers it for you.
- There is no RTT. You write a small **custom transport**: USB-Serial/JTAG or a UART.

1. What works

Item	Status
Chips	ESP32, ESP32-S2, ESP32-S3 (Xtensa LX6/LX7), with <code>TRAX_HW_PORT_XTENZA</code>
RISC-V ESP chips (ESP32-C2, C3, C5, C6, C61, H2, P4, ...)	Not supported. The port is written for Xtensa only, and the SDK has no port for RISC-V ESP chips
ESP-IDF	Tested with ESP-IDF 6.0.1 . Your ESP-IDF must have the ESP Trace component (<code>CONFIG_ESP_TRACE_LIB_EXTERNAL</code>). If <code>menuconfig</code> has no <i>ESP Trace Configuration</i> menu, your ESP-IDF is too old for this integration
Kernel	ESP-IDF FreeRTOS (the default, <code>CONFIG_FREERTOS_SMP</code> not set). Both cores are traced
Tested hardware	ESP32-S3 DevKitC with the USB-Serial/JTAG transport (§7.1). Classic ESP32 with a UART1 transport (§7.2). ESP32-S2 is supported by the port but not yet tested on hardware
Timestamps	<code>esp_timer</code> , 1 μ s resolution, shared by both cores
Kernel tracing	Task switches on CPU0 and CPU1, task create/delete, queues, semaphores, mutexes, the tick interrupt, heap events (Chapter 13)
Logs, variables, streams, markers, state machines, triggers	Same API as on any other target (Chapters 10–17)
Metadata	<code>TRAX_META_IN_FLASH</code> (tested). <code>TRAX_META_ELF_ONLY</code> is not tested on ESP-IDF
Crash capture	Not available on Xtensa (Chapter 20). Leave <code>TRAX_CFG_FAULT_ENABLE</code> at 0

License. The ESP32 integration works with every Traxcope license. In the Community Edition the usual limits apply: live streaming for 3 minutes per rolling hour, one connected probe, no new recordings (Chapter 04 §2).

2. Prerequisites

You need:

- Traxcope 2.0.0, installed with the TraxProbe SDK (Windows: **Full installation**). See Chapter 01.
- ESP-IDF 6.0.1 (or newer, with ESP Trace).
- An ESP32, ESP32-S2, or ESP32-S3 board.
- For the UART transport: a 3.3 V USB-UART adapter.

Install ESP-IDF on Windows

idf.py comes with ESP-IDF. It is not a separate program. Install Python and Git first.

1. Clone ESP-IDF at the release you want, with its submodules:

```
cd /d C:\esp
git clone -b v6.0.1 --recursive https://github.com/espressif/esp-idf.git
```

2. Install the tools for your chips (once):

```
cd C:\esp\esp-idf
install.bat esp32,esp32s3
```

3. In **every new** CMD window, activate the environment:

```
call "C:\esp\esp-idf\export.bat"
idf.py --version
```

export.bat sets up Python and the tool paths for that window only. You do not install again.

You can also use Espressif's Windows installer, the ESP-IDF VS Code extension, or Espressif-IDE. The commands below are the same in an activated ESP-IDF terminal.

Linux and macOS: follow Espressif's *Get Started* guide (install.sh, then . ./export.sh in each shell). Everything after this section is the same.

3. Create the project

```
cd /d C:\Workspace
idf.py create-project esp32_trax
cd esp32_trax
idf.py set-target esp32s3
```

- Use create-project, not create.
- Use your chip: esp32, esp32s2, or esp32s3.
- Run set-target **before** menuconfig. set-target regenerates sdkconfig and discards earlier menuconfig changes.

create-project creates main/esp32_trax.c and main/CMakeLists.txt.

4. Add the TraxProbe component

1. In the project root, create a folder named components.
2. Copy the whole SDK into it as components/TraxProbe. On a default Windows install:

```
xcopy /E /I "C:\Program Files\Embedya\Traxcope\sdk\TraxProbe" components\TraxProbe
```

On Ubuntu the SDK is under /opt/Embedya/Traxcope/sdk/TraxProbe (see Chapter 02).

3. Create two files in main/: trax_config.h (§5) and trax_transport.c (§7).

The result:

esp32_trax/	
CMakeLists.txt	project file (§6.1)
sdkconfig.defaults	trace settings (§6.3), optional but recommended
components/	
TraxProbe/	the SDK, unchanged – CMakeLists.txt is already there
main/	
CMakeLists.txt	(§6.2)
esp32_trax.c	your application (§8)
trax_config.h	(§5)
trax_transport.c	(§7)

Keep these names and this layout:

- The folder must be named exactly TraxProbe, directly under components/. Do not add another folder level.
- trax_config.h must be in main/. The component adds ../../main (seen from components/TraxProbe) to the include path, so the library and the FreeRTOS hooks find trax_config.h there.
- Do not edit files inside components/TraxProbe. After a Traxcope upgrade, copy the SDK again.
- On Windows, turn on file-name extensions in Explorer. The names must be trax_config.h and trax_transport.c, not trax_config.h.txt or c.example.

The component compiles the core, the Xtensa port (hw_port/Xtensa/src/trax_hw_port.c), the FreeRTOS port, and the ESP Trace adapter (sdk/esp/). It registers linker/esp/trax_meta.1f and is built WHOLE_ARCHIVE. It does **not** compile anything under transports/; your trax_transport.c provides the transport.

5. main/trax_config.h

There is no ESP-specific preset in config/templates/. Use this file as the starting point. It is complete for an ESP32-S3 or a classic ESP32 (both dual-core):

```
#ifndef APP_TRAX_CONFIG_H_
#define APP_TRAX_CONFIG_H_

/* --- Hardware port (MANDATORY) ----- */
#define TRAX_CFG_HW_PORT          TRAX_HW_PORT_XTENZA
#define TRAX_CFG_CORE_COUNT      2U      /* ESP32 / ESP32-S3. 1U on ESP32-S2 */

/* --- Timestamps (MANDATORY) -----
 * The Xtensa port reads esp_timer_get_time(): a 1 MHz counter shared by
 * both cores. The value below describes that counter, not the CPU clock. */
#define TRAX_CFG_TIMESTAMP_MODE    TRAX_TIMESTAMP_FREERUN
#define TRAX_CFG_TIMESTAMP_TIMER_DIR  TRAX_TIMER_DIR_UP
#define TRAX_CFG_TIMER_FREQ_HZ    1000000U

/* --- Transport: main/trax_transport.c ----- */
#define TRAX_CFG_TRANSPORT        TRAX_TRANSPORT_CUSTOM

/* --- Metadata: schema is sent at session start, no .elf needed ----- */
#define TRAX_CFG_META_STORAGE     TRAX_META_IN_FLASH

/* --- FreeRTOS kernel tracing ----- */
#define TRAX_CFG_RTOS_TYPE        TRAX_RTOS_FREERTOS
#define TRAX_CFG_FREERTOS_VERSION TRAX_FREERTOS_VERSION(10, 5, 1) /* use YOUR value, $5.1 */

/* TraxCtrl stack. On ESP-IDF this is in BYTES (StackType_t is uint8_t).
 * The library default of 256 is far too small here and crashes. */
#define TRAX_CFG_CTRL_TASK_STACK_SIZE 4096U

/* ESP-IDF starts many system tasks (IDLE0/1, ipc0/1, esp_timer, main, ...).
 * Raise the task table above the default 16 so none is missed. */
#define TRAX_CFG_MAX_RTOS_TASKS    32

/* --- Optional ----- */
#define TRAX_CFG_PROJECT_NAME      "esp32_trax"
#define TRAX_CFG_OUT_BUFFER_SIZE32 4096U /* 16 KB ring */
#define TRAX_CFG_LOG_COMPILE_LEVEL TRAX_LOG_LEVEL_TRACE

#endif /* APP_TRAX_CONFIG_H_ */
```

Notes:

Setting	Why
TRAX_CFG_TIMER_FREQ_HZ 1000000U	The port's counter is esp_timer at 1 MHz. It does not use the CPU cycle counter (CCOUNT), because CCOUNT is per core and not synchronized between the cores. Do not enter the CPU clock (160 or 240 MHz): every time in Traxcope would be wrong by that factor
TRAX_CFG_TIMESTAMP_MODE FREERUN	Do not set TRAX_CFG_TICK_COUNTER_PERIOD or TRAX_CFG_TIMESTAMP_TIMER_VAL. TRAX_CFG_TICK_BITS stays at its default (8)
TRAX_CFG_CORE_COUNT	2U on ESP32 and ESP32-S3, so events land on the CPU0 and CPU1 lanes. Use 1U on the single-core ESP32-S2, or when you build with CONFIG_FREERTOS_UNICORE=y
TRAX_CFG_CTRL_TASK_STACK_SIZE	In bytes on ESP-IDF. The default, 256, is meant as 256 words on Cortex-M; on ESP-IDF it gives a 256-byte stack and corrupts memory
TRAX_LOG_COMPILE_LEVEL	No TRAX_CFG_ prefix. TRAX_CFG_LOG_COMPILE_LEVEL is silently ignored
TRAX_CFG_META_STORAGE	TRAX_META_IN_FLASH is the tested mode on ESP-IDF

You do not call trax_timestamp_tick() or trax_timestamp_poll(). With FreeRTOS tracing on, the ESP-IDF tick hook calls it.

TID ranges, buffer sizing, and the other options are the same as on any target: Chapter 02 §2 and §7.

5.1 Find the FreeRTOS version

TRAX_CFG_FREERTOS_VERSION is mandatory with TRAX_RTOS_FREERTOS. It is the **FreeRTOS kernel** version, not the ESP-IDF version. In an activated ESP-IDF CMD window:

```
findstr /C:"tskKERNEL_VERSION_NUMBER" "%IDF_PATH%\components\freertos\FreeRTOS-
Kernel\include\freertos\task.h"
```

The output looks like:

```
#define tskKERNEL_VERSION_NUMBER "V10.5.1"
```

Write the three numbers into the macro: TRAX_FREERTOS_VERSION(10, 5, 1).

- Use the value your ESP-IDF prints. 10, 5, 1 above is only an example.
- If you enabled CONFIG_FREERTOS_SMP, the kernel is under FreeRTOS-Kernel-SMP instead. This configuration is not tested.
- A recursive search (findstr /S ... "%IDF_PATH%\components\freertos*.h") finds both kernels. Read the line from FreeRTOS-Kernel.
- A missing macro, or a version below V10.2.0, stops the build with an #error.

6. Build files and menuconfig

6.1 Root CMakeLists.txt

Replace the generated file with:

```
cmake_minimum_required(VERSION 3.22)

include($ENV{IDF_PATH}/tools/cmake/project.cmake)
# Build only main and what it depends on.
idf_build_set_property(MINIMAL_BUILD ON)
project(esp32_trax)

# FreeRTOS trace hooks: give every file that includes FreeRTOSConfig.h
# (the kernel included) the TraxProbe include paths, through esp_trace.
if(CONFIG_ESP_TRACE_LIB_EXTERNAL)
    idf_component_get_property(esp_trace_lib esp_trace COMPONENT_LIB)
    idf_component_get_property(traxprobe_lib TraxProbe COMPONENT_LIB)
    target_link_libraries(${esp_trace_lib} INTERFACE ${traxprobe_lib})
endif()
```

Why each line is there:

- **The target_link_libraries block.** ESP-IDF's FreeRTOSConfig.h includes TraxProbe's sdk/esp/esp_trace_freertos_impl.h. Without this link the kernel cannot find that header, trax.h, or trax_config.h, and the build fails with esp_trace_freertos_impl.h: No such file or directory. The block must come **after** project(), because the component targets exist only then. It cannot live in the TraxProbe component itself: components are configured in name order, and TraxProbe comes before esp_trace.
- **idf_component_get_property.** It looks up the real library targets. Do not hard-code internal target names.
- **MINIMAL_BUILD ON.** Once the trace hooks put TraxProbe's headers into every file that includes FreeRTOSConfig.h, TraxProbe's argument-counting macro PP_NARG can clash with a macro of the same name in ESP-IDF's test components (unity, cmock), and ESP-IDF treats the warning as an error. A minimal build compiles only main and its dependencies, so those test components stay out of the build.

6.2 main/CMakeLists.txt

For the USB-Serial/JTAG transport (ESP32-S3):

```
idf_component_register(SRCS "esp32_trax.c"
                        "trax_transport.c"
                        INCLUDE_DIRS "."
                        REQUIRES TraxProbe esp_timer esp_trace esp_driver_usb_serial_jtag
                        WHOLE_ARCHIVE)
```

For the UART transport, replace esp_driver_usb_serial_jtag with esp_driver_uart.

- Put trax_transport.c in main. With MINIMAL_BUILD, a separate component is built only if main lists it in REQUIRES; if it is left out, the link fails with undefined reference to 'trax_transport_...'.
- WHOLE_ARCHIVE keeps the transport functions in the link. TraxProbe calls them; main does not.

6.3 menuconfig

Set two options:

Option	Symbol	Value
Component config → FreeRTOS → Kernel → configUSE_TRACE_FACILITY	CONFIG_FREERTOS_USE_TRACE_FACILITY	on
Component config → ESP Trace Configuration → Trace library	CONFIG_ESP_TRACE_LIB_EXTERNAL	External library

Leave **Trace transport** at **None** (CONFIG_ESP_TRACE_TRANSPORT_NONE). TraxProbe sends its data through your own transport, not through ESP Trace. Do not pick the *External* transport.

1. Run idf.py menuconfig.
2. Set the options. If a menu label differs in your ESP-IDF, press / and search for the symbol name.
3. Save and exit.

To keep these settings when you run `set-target` again, also put them in `sdkconfig.defaults` in the project root:

```
CONFIG_FREERTOS_USE_TRACE_FACILITY=y
CONFIG_ESP_TRACE_LIB_EXTERNAL=y
```

What the two options do:

- `configUSE_TRACE_FACILITY` adds the task and queue numbers that the trace hooks use. It also lets TraxProbe list the tasks that already run when `trax_init()` is called, so the idle tasks get their names. Without it the build stops with an `#error`.
- *External library* makes ESP-IDF include TraxProbe's FreeRTOS hooks. The TraxProbe component also registers the small ESP Trace stubs that ESP-IDF looks up at boot.

7. Transport

The transport is four functions in `main/trax_transport.c`. The contract is in Chapter 02 §3.2. In short:

Function	Rule
<code>trax_transport_init()</code>	Install the driver. Return 0 on success
<code>trax_transport_write()</code>	All-or-nothing: return size (every byte queued) or 0 (nothing queued, try again). Never a partial count
<code>trax_transport_read()</code>	Non-blocking. Return what is there now, often 0. Never wait for data
<code>trax_transport_clear_tx()</code>	Called when a session restarts. Empty your own software TX buffer; never touch RX. Bytes already handed to the driver cannot be recalled: then the body is empty

All four are required. A missing one is a link error.

These functions run in the TraxCtrl task. Do not call `printf` or `ESP_LOG*` in them, and do not wait in `read()`.

Keep the console and the trace link on different ports. Console text on the trace port corrupts the binary stream. In both transports below the console stays on UART0, the port you flash and monitor with.

7.1 USB-Serial/JTAG (ESP32-S3)

The ESP32-S3 has a built-in USB-Serial/JTAG controller (the **USB** connector on the ESP32-S3-DevKitC). TraxProbe uses it for the trace data. The console stays on UART0 (the **UART** connector). No extra hardware is needed. ESP32 and ESP32-S2 have no USB-Serial/JTAG controller; use §7.2 there.

Make sure the console does not also write to USB-Serial/JTAG. In `sdkconfig.defaults` (or `menuconfig`, search `ESP_CONSOLE_SECONDARY`):

```
CONFIG_ESP_CONSOLE_SECONDARY_NONE=y
```

`main/trax_transport.c`:

```
#include <trax_transport.h>

#include "driver/usb_serial_jtag.h"
#include "freertos/FreeRTOS.h"

#include <string.h>

#define USJ_TX_RING_BYTES    (2048U)
#define USJ_RX_RING_BYTES    (1024U)
#define RX_STAGE_BYTES       (512U)

/* Host-command prefetch buffer. */
static uint8_t s_rx[RX_STAGE_BYTES];
static size_t s_rx_len = 0;
static size_t s_rx_rd = 0;

/* Move whatever the driver has into s_rx. Never waits. */
static void rx_pump(void)
{
    if (s_rx_rd == s_rx_len) {
        s_rx_rd = 0;
        s_rx_len = 0;
    }
    if (s_rx_len < RX_STAGE_BYTES) {
        int n = usb_serial_jtag_read_bytes(&s_rx[s_rx_len],
                                           RX_STAGE_BYTES - s_rx_len,
                                           0 /* non-blocking */);

        if (n > 0) {
            s_rx_len += (size_t)n;
        }
    }
}

int trax_transport_init(void)
{
    usb_serial_jtag_driver_config_t cfg = {
        .tx_buffer_size = USJ_TX_RING_BYTES,
        .rx_buffer_size = USJ_RX_RING_BYTES,
    };
    return (usb_serial_jtag_driver_install(&cfg) == ESP_OK) ? 0 : -1;
}

size_t trax_transport_write(const void *p_src, size_t size)
{
    const uint8_t *p = (const uint8_t *)p_src;
    int w = usb_serial_jtag_write_bytes(p, size, 0);
    if (w <= 0) {
        return 0U; /* nothing queued: core retries */
    }
    if ((size_t)w == size) {
        return size;
    }
    /* The driver took only a prefix: finish it, never return a partial count. */
    p += w;
    size_t remaining = size - (size_t)w;
    while (remaining > 0U) {
        w = usb_serial_jtag_write_bytes(p, remaining, pdMS_TO_TICKS(20));
        if (w > 0) {
            p += w;
            remaining -= (size_t)w;
        }
    }
    return size;
}

size_t trax_transport_read(void *p_dst, size_t size)
```

```
{
    rx_pump();

    size_t avail = s_rx_len - s_rx_rd;
    size_t n = (size < avail) ? size : avail;
    if (n > 0U) {
        memcpy(p_dst, &s_rx[s_rx_rd], n);
        s_rx_rd += n;
    }
    return n;
}

void trax_transport_clear_tx(void)
{
    /* No software TX buffer of our own. Bytes already in the driver
     * cannot be recalled; the host treats them as pre-session data. */
}
```

Ports on the ESP32-S3-DevKitC:

Connector	Used for	Traxcope
UART	Flash, console (idf.py monitor)	—
USB	TraxProbe data	UART transport, this COM port

7.2 UART1 with a USB-UART adapter (classic ESP32 and others)

Use this on a classic ESP32 or an ESP32-S2, or on any chip where you want a plain UART. UART0 stays the console. UART1 carries the trace data to a separate 3.3 V USB-UART adapter.

Wiring (pins as in the code below):

ESP32	Adapter
GPIO4 (UART1 TX)	RX
GPIO5 (UART1 RX)	TX
GND	GND

- Use a **3.3 V** adapter. 5 V logic can damage the chip.
- Cross TX and RX. Connect GND.
- UART1's default pins on the classic ESP32 are wired to the SPI flash. Always set the pins with `uart_set_pin()`.
- Pin choice on the classic ESP32: do not use GPIO6–11 (flash). Avoid GPIO0, GPIO2, GPIO12 and GPIO15, which are strapping pins. GPIO5 is also a strapping pin, but its boot default is high, and an idle UART line from the adapter is high, so it works as RX. On WROVER modules GPIO16 and GPIO17 are used by PSRAM. GPIO34–39 are input-only (RX only).
- If you change pins, change the wiring and the code together.

main/trax_transport.c:

```
#include <trax_transport.h>

#include "driver/uart.h"

#define TRAX_UART_PORT          UART_NUM_1
#define TRAX_UART_TX_PIN        4          /* ESP32 TX -> adapter RX */
#define TRAX_UART_RX_PIN        5          /* ESP32 RX <- adapter TX */
#define TRAX_UART_BAUD          921600
#define TRAX_UART_TX_RING       4096       /* bytes */
#define TRAX_UART_RX_RING       1024       /* bytes, must be > 128 */

int trax_transport_init(void)
{
    const uart_config_t cfg = {
        .baud_rate    = TRAX_UART_BAUD,
        .data_bits    = UART_DATA_8_BITS,
        .parity        = UART_PARITY_DISABLE,
        .stop_bits     = UART_STOP_BITS_1,
        .flow_ctrl     = UART_HW_FLOWCTRL_DISABLE,
        .source_clk    = UART_SCLK_DEFAULT,
    };

    if (uart_driver_install(TRAX_UART_PORT, TRAX_UART_RX_RING,
                           TRAX_UART_TX_RING, 0, NULL, 0) != ESP_OK) {
        return -1;
    }
    if (uart_param_config(TRAX_UART_PORT, &cfg) != ESP_OK) {
        return -1;
    }
    if (uart_set_pin(TRAX_UART_PORT, TRAX_UART_TX_PIN, TRAX_UART_RX_PIN,
                    UART_PIN_NO_CHANGE, UART_PIN_NO_CHANGE) != ESP_OK) {
        return -1;
    }
    return 0;
}

size_t trax_transport_write(const void *p_src, size_t size)
{
    /* uart_write_bytes() returns only after the whole batch is in the
     * driver's TX ring (it waits while the ring is full), so this is
     * all-or-nothing. An error returns -1: report "nothing queued". */
    int n = uart_write_bytes(TRAX_UART_PORT, p_src, size);
    return (n == (int)size) ? size : 0U;
}

size_t trax_transport_read(void *p_dst, size_t size)
{
    /* Timeout 0: return what has arrived, never wait. */
    int n = uart_read_bytes(TRAX_UART_PORT, p_dst, (uint32_t)size, 0);
    return (n > 0) ? (size_t)n : 0U;
}

void trax_transport_clear_tx(void)
{
    /* Empty on purpose. Bytes in the UART driver cannot be recalled.
     * Do NOT call uart_flush() or uart_flush_input() here: they clear
     * RX (host commands), not TX. */
}
```

921600 baud carries about 92 KB/s. That is enough for kernel tracing plus logs and a few variables (Chapter 02 §7). If your adapter is not reliable at that rate, lower `TRAX_UART_BAUD` and use the same value in Traxcope.

7.3 Other links

Any link works if you write the four functions for it. TCP over Wi-Fi needs its own custom transport over lwIP sockets: the SDK's built-in TCP transport (TRAX_TRANSPORT_TCP) is meant for Linux and VxWorks and is not built by the ESP-IDF component. In Traxcope, pair a custom TCP transport with the **TCP** transport type.

8. Application code

On ESP-IDF the scheduler is already running when `app_main()` starts, and `app_main()` runs in a task. Call `trax_init()` there. Under FreeRTOS, TraxProbe's **TraxCtrl** task drains the buffer and handles host commands. Do not call `trax_process()` yourself.

main/esp32_trax.c:

```
#include <stdint.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "trax.h"

enum { LOG_TID_HELLO    = TRAX_TID_RANGE_LOG_USER_START };
enum { VAR_TID_COUNTER = TRAX_TID_RANGE_VAR_USER_START };

/* Optional: some work on CPU1, so the CPU1 lane is not only idle. */
static void core1_worker(void *arg)
{
    (void)arg;
    volatile uint32_t sink = 0;
    for (;;) {
        for (uint32_t i = 0; i < 200000U; ++i) {
            sink += i;
        }
        vTaskDelay(pdMS_TO_TICKS(50));
    }
}

void app_main(void)
{
    trax_init();
    trax_wait_session_started(); /* optional: wait until Traxcope starts streaming */

    xTaskCreatePinnedToCore(core1_worker, "core1_work", 2048, NULL, 3, NULL, 1);

    uint32_t counter = 0;
    for (;;) {
        counter++;
        TRAX_VAR_SET(VAR_TID_COUNTER, counter);
        if ((counter % 20U) == 0U) {
            TRAX_LOG_INFO(LOG_TID_HELLO, "demo", "counter=%u", (unsigned)counter);
        }
        vTaskDelay(pdMS_TO_TICKS(50));
    }
}
```

This sends one variable sample about every 50 ms and one log line about every second. Remove `trax_wait_session_started()` if the firmware must run without Traxcope connected. For real projects, keep TIDs in small headers (Chapter 02 §2.4).

9. Build and flash

From the project root, in an activated ESP-IDF terminal:

```
idf.py build
idf.py -p COM3 flash monitor
```

- COM3 is the **console/flash** port (the board's UART connector or on-board USB-UART bridge). It is not the trace port.
- Leave the monitor with Ctrl+] . You can keep it open while Traxcope uses the trace port: they are different ports.
- Rebuild and flash after any change to `trax_config.h`, the transport, or `sdkconfig`.

10. Connect in Traxcope

1. Plug in the trace port: the ESP32-S3 **USB** connector (§7.1), or the USB-UART adapter (§7.2). Note its COM port in Device Manager (Linux: `/dev/ttyACM*` or `/dev/ttyUSB*`).
2. In Traxcope, open **Probes** → **Add**.
3. **General** tab: give the probe a name. The **ELF File** is optional: with `TRAX_META_IN_FLASH` the firmware sends its schema at session start.
4. **Transport** tab:

Field	USB-Serial/JTAG (§7.1)	UART1 adapter (§7.2)
Type	UART	UART
Port	The ESP32-S3 USB port	The adapter's port
Baud Rate	Any value (USB link, the rate is not used)	921600 (same as <code>TRAX_UART_BAUD</code>)
Data Bits / Stop Bits / Parity	8 / 1 / None	8 / 1 / None
Direction	Bidirectional	Bidirectional

Click ↻ to refresh the port list. Click **Test** to check the link.

5. Save the probe. Click **Connect**, then **Start Stream**.

The firmware info card shows the project name from `TRAX_CFG_PROJECT_NAME`. The general probe steps are in Chapter 01 Step 5 and Chapter 03 §2.2.

11. What you see

Open **Toolbox** → **Trace**:

- The **CPU** group has a **CPU0** and a **CPU1** lane. Each shows what ran on that core.
- Task lanes show ESP-IDF's own tasks (for example `IDLE0`, `IDLE1`, `ipc0`, `ipc1`, `esp_timer`, `main`), your tasks, and **TraxCtrl**.
- The tick interrupt has its own ISR lane.
- Idle time on each core is your CPU headroom.

TraxCtrl should take a small share of CPU time. If it takes most of a core, the setup is wrong; see §12.

Open **Log** for the demo lines and **Scope** for `VAR_TID_COUNTER`. Everything else in Traxcope works as on any other target: Chapter 13 for the Trace View, Chapter 10 for logs, Chapter 11 for variables.

12. Troubleshooting

Symptom	Cause and fix
Build: esp_trace_freertos_impl.h: No such file or directory	The target_link_libraries block is missing from the root CMakeLists.txt, or it is before project() (§6.1)
Build: trax_config.h: No such file or directory	trax_config.h is not in main/, or the SDK is not at components/TraxProbe (§4)
Build: PP_NARG redefined (warning treated as error)	unity/cmock are in the build. Add idf_build_set_property(MINIMAL_BUILD ON) before project() (§6.1). If the project needs another component that pulls in unity, set(EXCLUDE_COMPONENTS "unity" "cmock") before the include(...project.cmake) line has also worked, but only if nothing else needs those components
Build: #error asking for TRAX_CFG_FREERTOS_VERSION, or "supports FreeRTOS V10.2.0 and newer"	Set the macro from task.h (§5.1)
Build: #error "requires configUSE_TRACE_FACILITY == 1"	Turn on CONFIG_FREERTOS_USE_TRACE_FACILITY (§6.3)
Link: undefined reference to 'trax_transport_...'	One of the four functions is missing (often trax_transport_clear_tx), or trax_transport.c is not in main/CMakeLists.txt SRCS (§6.2)
Link: orphan section .trax_...	You built the TraxProbe sources outside the supplied component, so linker/esp/trax_meta.lf is not registered. Use components/TraxProbe as shipped. If the section name ends in _elf, switch to TRAX_META_IN_FLASH (ELF-only is not tested on ESP-IDF)
Boot abort: esp_trace_core: Encoder 'ext' or transport 'none' not found	ESP Trace transport is not None , or the TraxProbe component is not in the build. Set Trace transport = None (§6.3)
Crash (LoadProhibited, often in the tick path) soon after trax_init()	TRAX_CFG_CTRL_TASK_STACK_SIZE not set. On ESP-IDF it is in bytes; set 4096U (§5)
Connect works, no data or garbage	Console and trace share a port. On ESP32-S3 set CONFIG_ESP_CONSOLE_SECONDARY_NONE=y (§7.1). Do not open the trace port in idf.py monitor. Remove any printf from the transport
Garbage on the UART adapter	Baud rate in Traxcope differs from TRAX_UART_BAUD; TX/RX not crossed; GND not connected; 5 V adapter
Times or durations wrong by a large factor	TRAX_CFG_TIMER_FREQ_HZ is not 1000000U. It describes esp_timer, not the CPU clock
All events on CPU0, CPU1 lane empty	TRAX_CFG_CORE_COUNT not set to 2U on a dual-core chip
Some tasks show as raw handles	More tasks than TRAX_CFG_MAX_RTOS_TASKS. Raise it (§5)
TraxCtrl takes most of the CPU	Check, in order: TRAX_CFG_TIMER_FREQ_HZ is 1000000U; read() does not wait (timeout 0) and has no printf; write() does not spin. Then compare the traffic with the link rate and the ring size (Chapter 02 §7)
Gaps in the timeline under load	The link is slower than the trace traffic. Raise TRAX_CFG_OUT_BUFFER_SIZE32, use a faster link, or reduce logs and variables
No ESP Trace Configuration menu in menuconfig	Your ESP-IDF has no ESP Trace component. Use ESP-IDF 6.0.1 or newer