Debugging

IDF TEAM 2018/09

Overview

- Overview
- Debugging with GDB + OpenOCD
- app_trace module in ESP-IDF
- Tracing with SystemView
- Coverage profiling with GCOV
- Flash download over JTAG

Overview of debugging techniques

Debugging

- Panic backtrace
- GDB stub
- Core dump
- OpenOCD + GDB

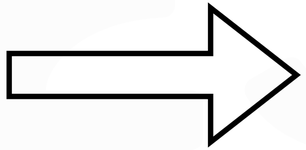
Analysis

- printf / ESP_LOG
- GPIO tracing
- CPU trace
- app_trace
- SystemView
- GCOV

Overview: panic and exception handling

Panic:
Assert, Abort, Cache access error interrupt, Brownout interrupt, Stack overflow, etc.

Unhandled exception:
Access to invalid memory address, Unaligned access, Illegal instruction, etc.



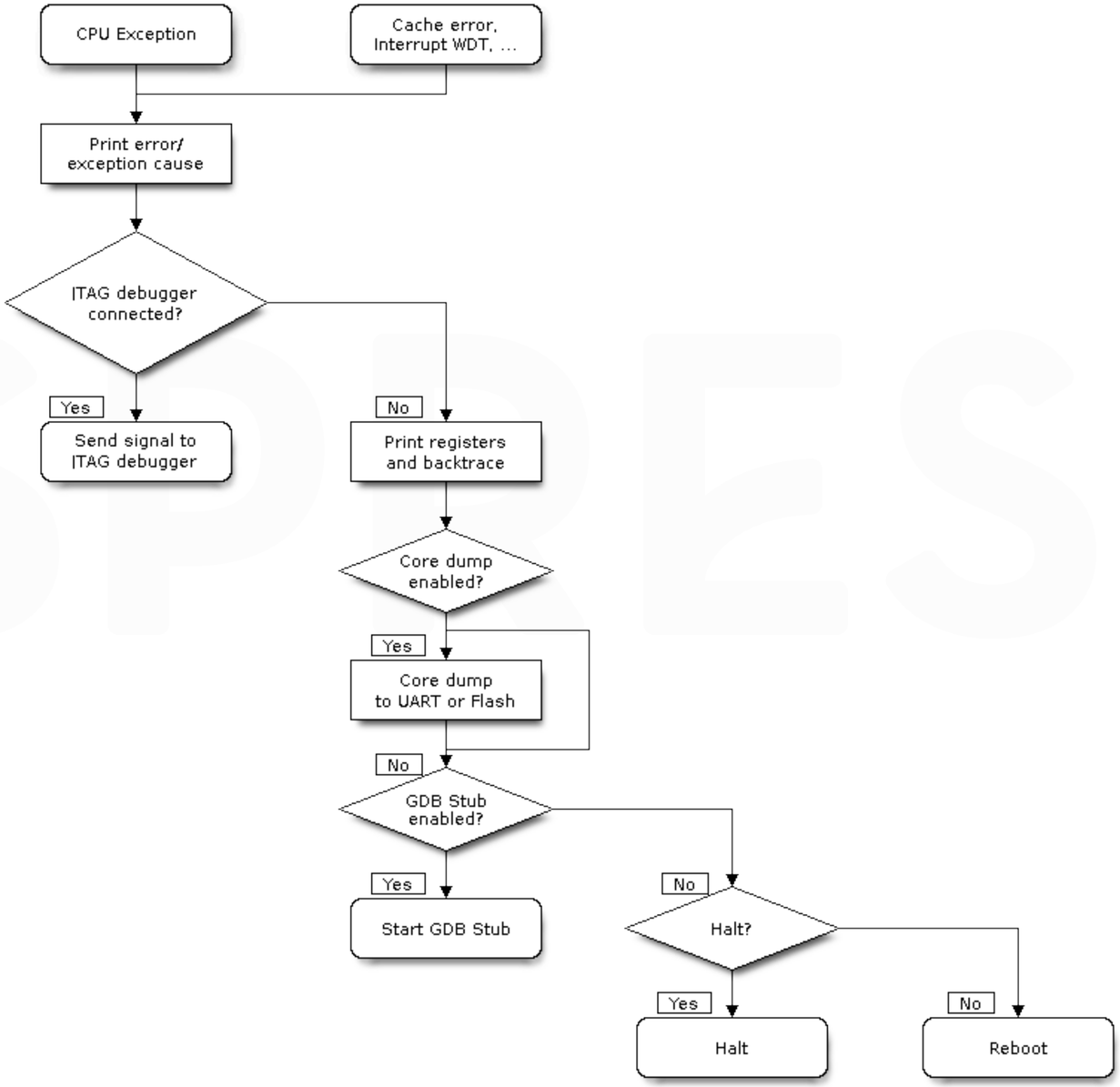
Register dump + **Backtrace**

GDB Stub

Core dump
(UART or Flash)

OCD/JTAG

Panic handler flow



Register dump / backtrace

Enabled by default. Gives the following information:

- what happened
- on which CPU
- where in the code
- what was the call stack.

```
Guru Meditation Error of type StoreProhibited occurred on core  0. Exception was unhandled
Register dump:
PC      : 0x400df56b  PS      : 0x00060a30  A0      : 0x800dc4f8  A1      : 0x3ffbccf0
A2      : 0x00000000  A3      : 0x3ffb1124  A4      : 0x00000001  A5      : 0x3ffbcd30
A6      : 0x3ffb2bf8  A7      : 0x3ffbcd30  A8      : 0x00000000  A9      : 0x00000001
A10     : 0x0005bce8  A11     : 0x00000000  A12     : 0x0005bce8  A13     : 0x00000000
A14     : 0x00000012  A15     : 0xff000000  SAR     : 0x00000004  EXCCAUSE: 0x0000001d
EXCVADDR: 0x00000000  LBEG    : 0x400014fd  LEND    : 0x4000150d  LCOUNT  : 0xfffffffb

Backtrace: 0x400df56b:0x3ffbccf0 0x400dc4f5:0x3ffbcd10 0x400dbf76:0x3ffbcd40
0x400dbfa1:0x3ffbcd60 0x400dc09b:0x3ffbcd80 0x400dc2ae:0x3ffbcda0 0x400d1b35:0x3ffbcec0
```

Connect and Innovate
连接与创新

Register dump / backtrace

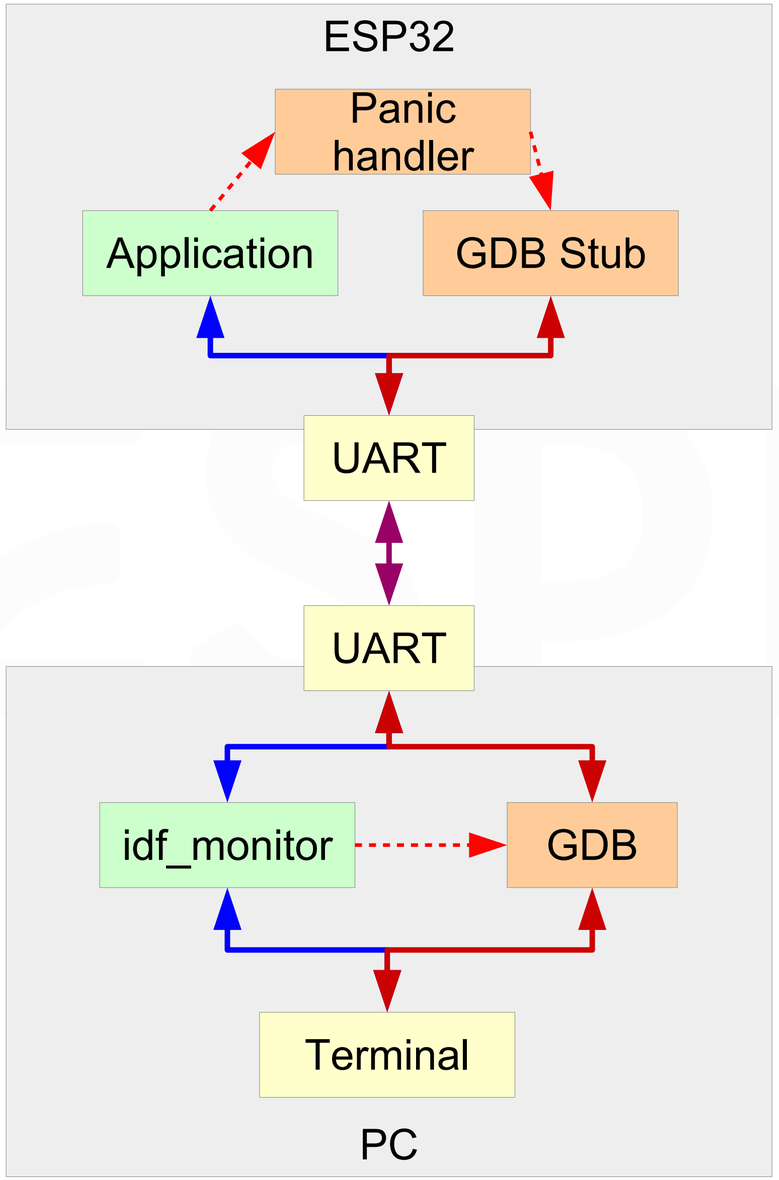
- idf_monitor.py tool (‘make monitor’/‘idf.py monitor’) helps generate readable backtrace
- Needs program elf file to generate backtrace

```
Guru Meditation Error of type StoreProhibited occurred on core  0. Exception was unhandled.
Register dump:
PC      : 0x400df56b  PS      : 0x00060a30  A0      : 0x800dc4f8  A1      : 0x3ffbccf0
0x400df56b: test_func_4 at tools/unit-test-app/main/test/./test_panic.c:6

A2      : 0x00000000  A3      : 0x3ffb1124  A4      : 0x00000001  A5      : 0x3ffbcd30
A6      : 0x3ffb2bf8  A7      : 0x3ffbcd30  A8      : 0x00000000  A9      : 0x00000001
A10     : 0x0005bce8  A11     : 0x00000000  A12     : 0x0005bce8  A13     : 0x00000000
A14     : 0x00000012  A15     : 0xff000000  SAR     : 0x00000004  EXCCAUSE: 0x0000001d
EXCVADDR: 0x00000000  LBEG    : 0x400014fd  LEND    : 0x4000150d  LCOUNT  : 0xfffffffb

Backtrace: 0x400df56b:0x3ffbccf0 0x400dc4f5:0x3ffbcd10 0x400dbf76:0x3ffbcd40 0x400dbfa1:0x3ffbcd60 0x400dc09b:
0x3ffbcd80 0x400dc2ae:0x3ffbcda0 0x400d1b35:0x3ffbcec0
0x400df56b: test_func_4 at tools/unit-test-app/main/test/./test_panic.c:6
0x400dc4f5: UnityDefaultTestRun at tools/unit-test-app/components/unity/./unity.c:1251
0x400dbf76: unity_run_single_test at tools/unit-test-app/components/unity/./unity_platform.c:121
0x400dbfa1: unity_run_single_test_by_index at tools/unit-test-app/components/unity/./unity_platform.c:132
0x400dc09b: unity_run_single_test_by_index_parse at tools/unit-test-app/components/unity/./unity_platform.c:148
0x400dc2ae: unity_run_menu at tools/unit-test-app/components/unity/./unity_platform.c:275
0x400d1b35: unityTask at tools/unit-test-app/main/./app_main.c:10
```

GDB Stub



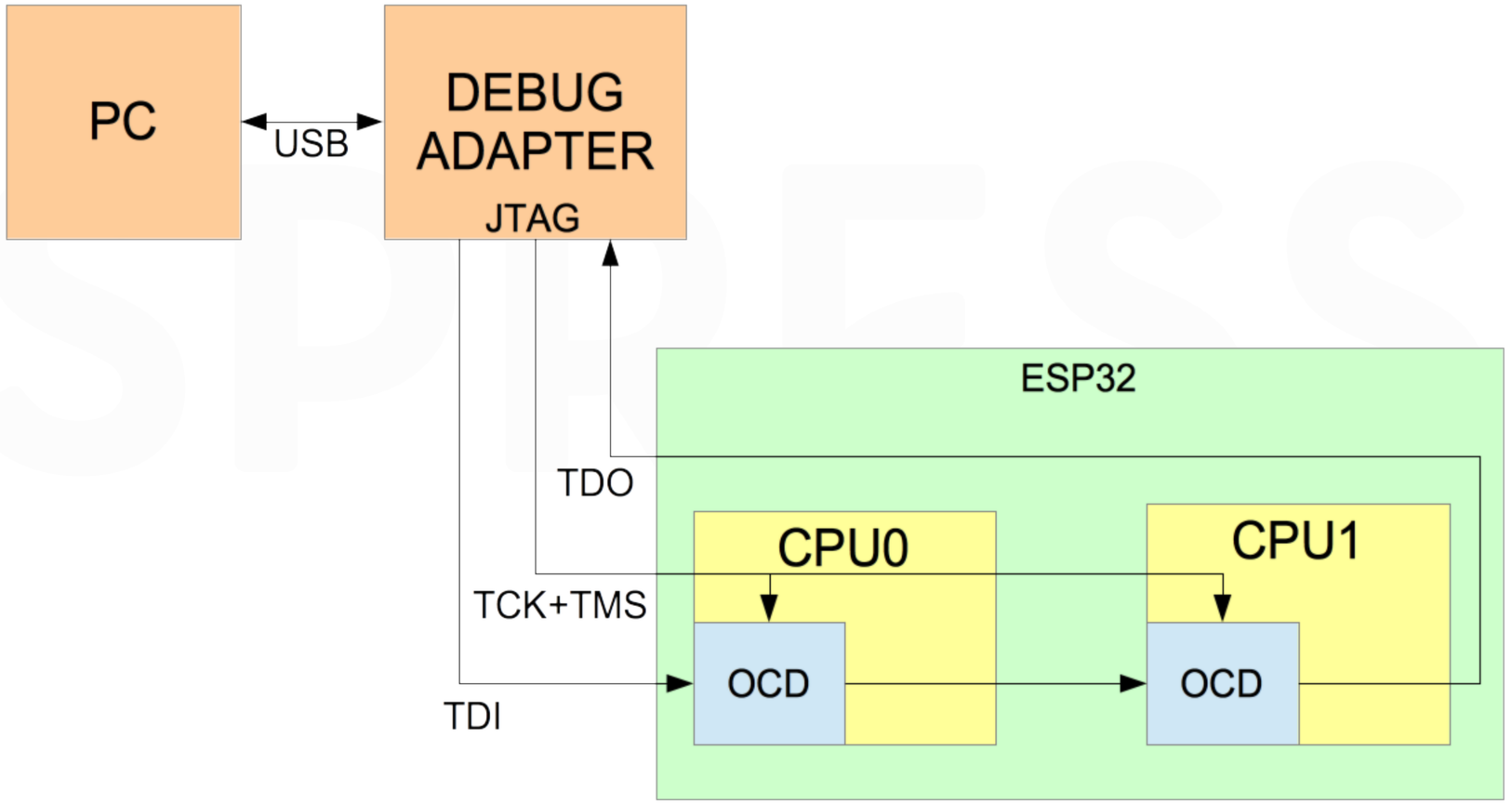
- GDB Remote Protocol server
- Enabled by CONFIG_ESP32_PANIC_GDBSTUB
- Started by panic handler
- idf_monitor can automatically launch GDB
- GDB Stub talks to GDB over UART, lets GDB inspect memory and registers, and build a stack trace
- GDB console commands should be used
- Can not use breakpoints or continue execution, can not switch between tasks

Core dump

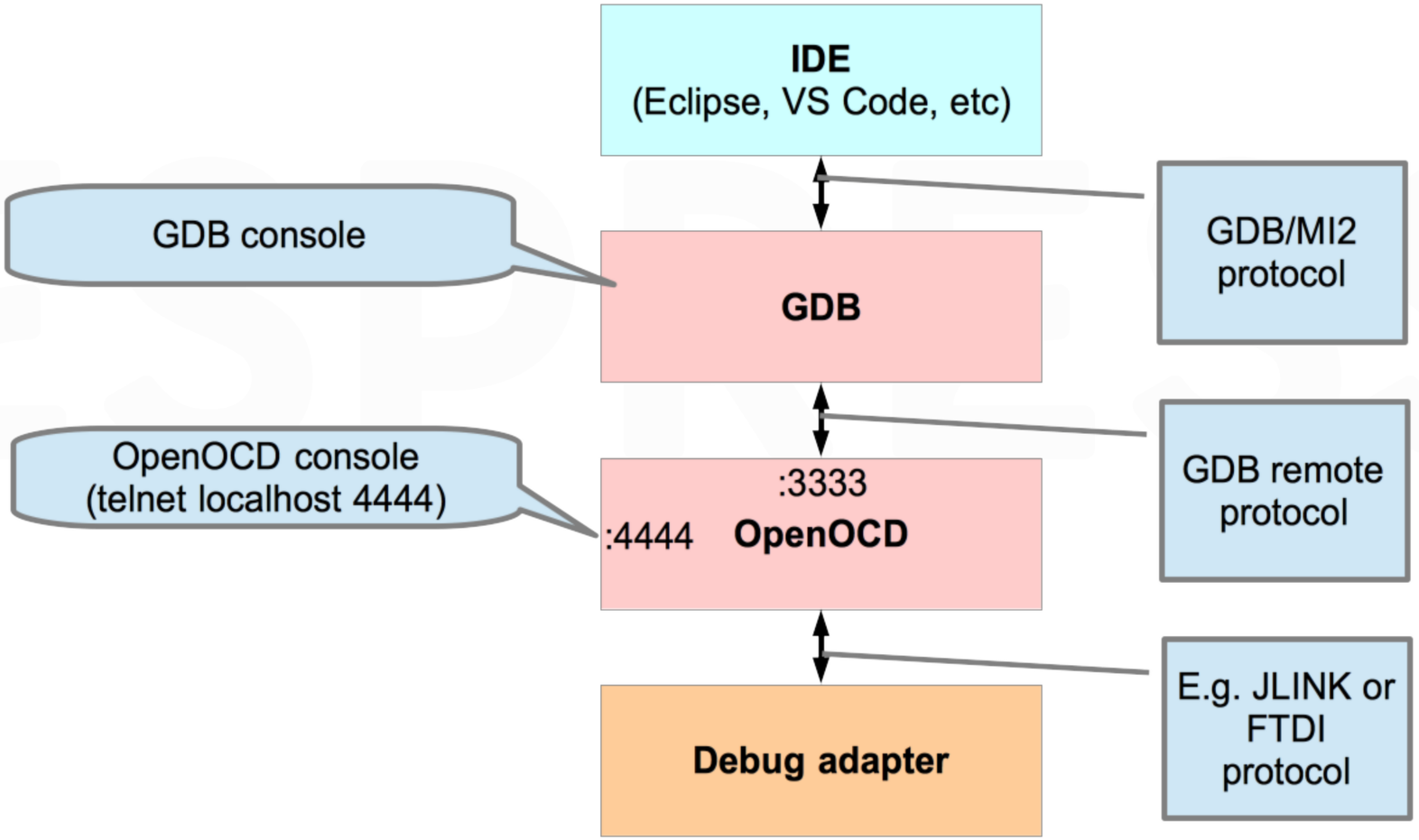
- Can be enabled in menuconfig
- Core dump module saves stacks and registers of all tasks
- Can write to Flash partition or UART
- espcoredump.py tool can decode core dump from Flash or UART.
- Launches GDB or prints backtrace.

https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/core_dump.html

JTAG/OCD: system diagram



JTAG/OCD: software diagram

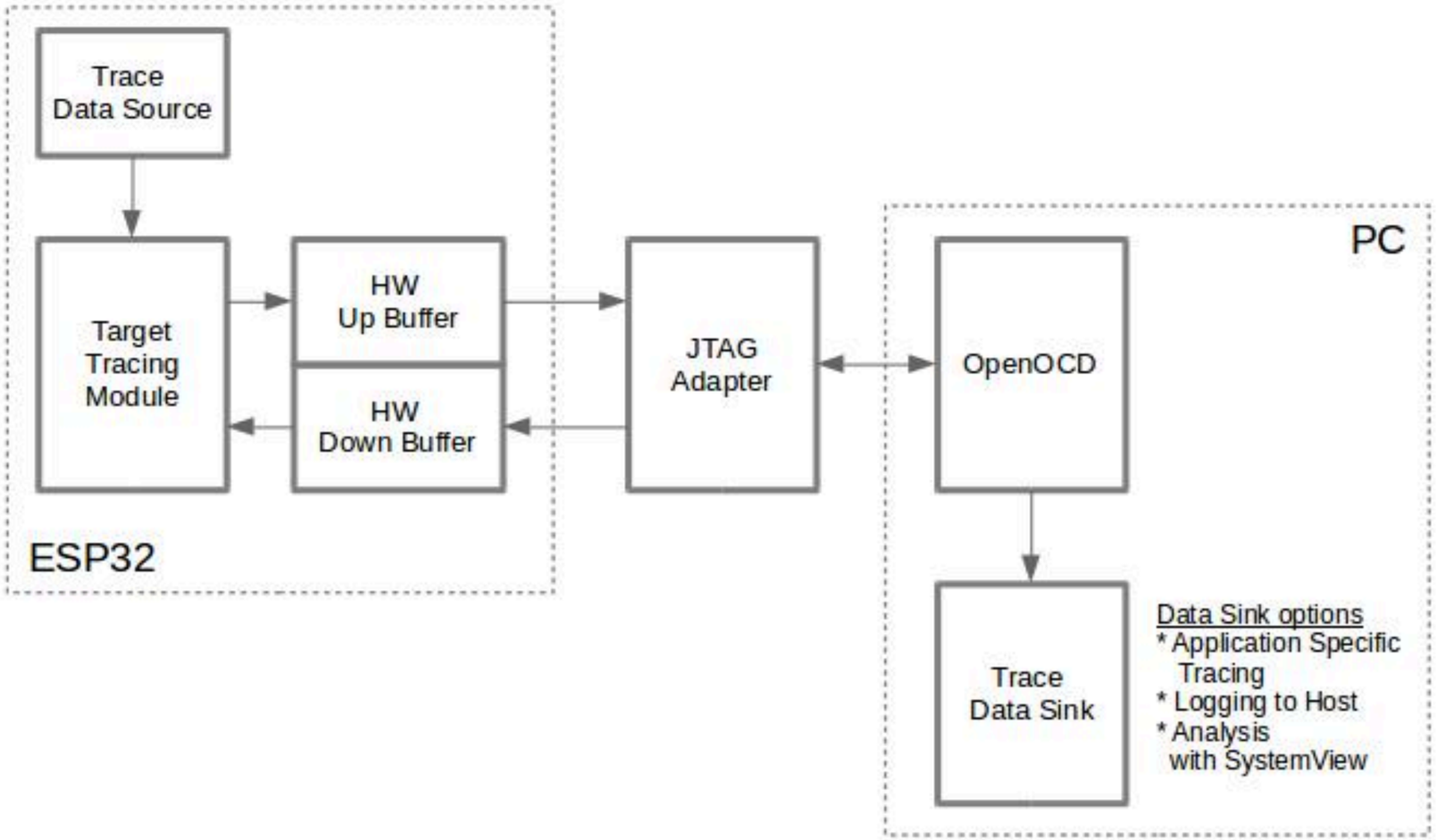


JTAG Debugging Checklist

1. Get a JTAG capable dev board (e.g. WROVER-KIT), install drivers
2. Download and extract OpenOCD
3. (Optional) Set up Eclipse IDE
4. Connect the board
5. Start OpenOCD
6. Start debugging in IDE or directly with GDB

Follow the guide here:
https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/jtag-debugging/index.html
Latest version can be downloaded here:
https://gitlab.espressif.cn:6688/idf/openocd-esp32/pipelines
Generating debug logs:
https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/jtag-debugging/tips-and-quirks.html#jtag-debugging-tip-reporting-issues
Report issues here:
https://gitlab.espressif.cn:6688/idf/openocd-esp32/issues

app_trace module



SystemView tracing

- If tracing is enabled, FreeRTOS calls tracing macros
- These macros send use app_trace module to send events over JTAG to the PC
- OpenOCD on PC saves events into .SVdat file
- SystemView displays the contents of SVdat file

Tutorial: https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/app_trace.html#system-behaviour-analysis-with-segger-systemview

Coverage profiling with GCOV

- When coverage is enabled, GCC inserts counters into the program
- When program is done, counter values are sent over JTAG to the PC using app_trace module
- Coverage files on PC are parsed by a tool such as lcov

Instructions: example/system/gcov/README.md