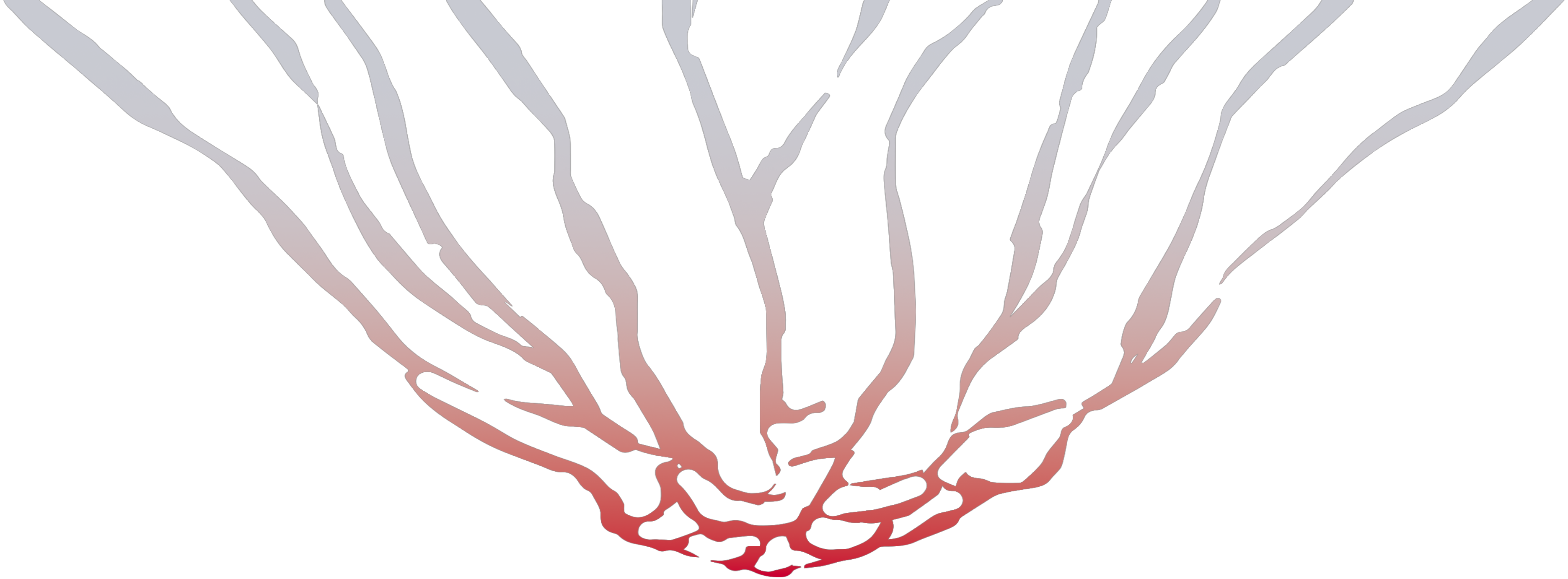


ESPRESSIF

TRAINING

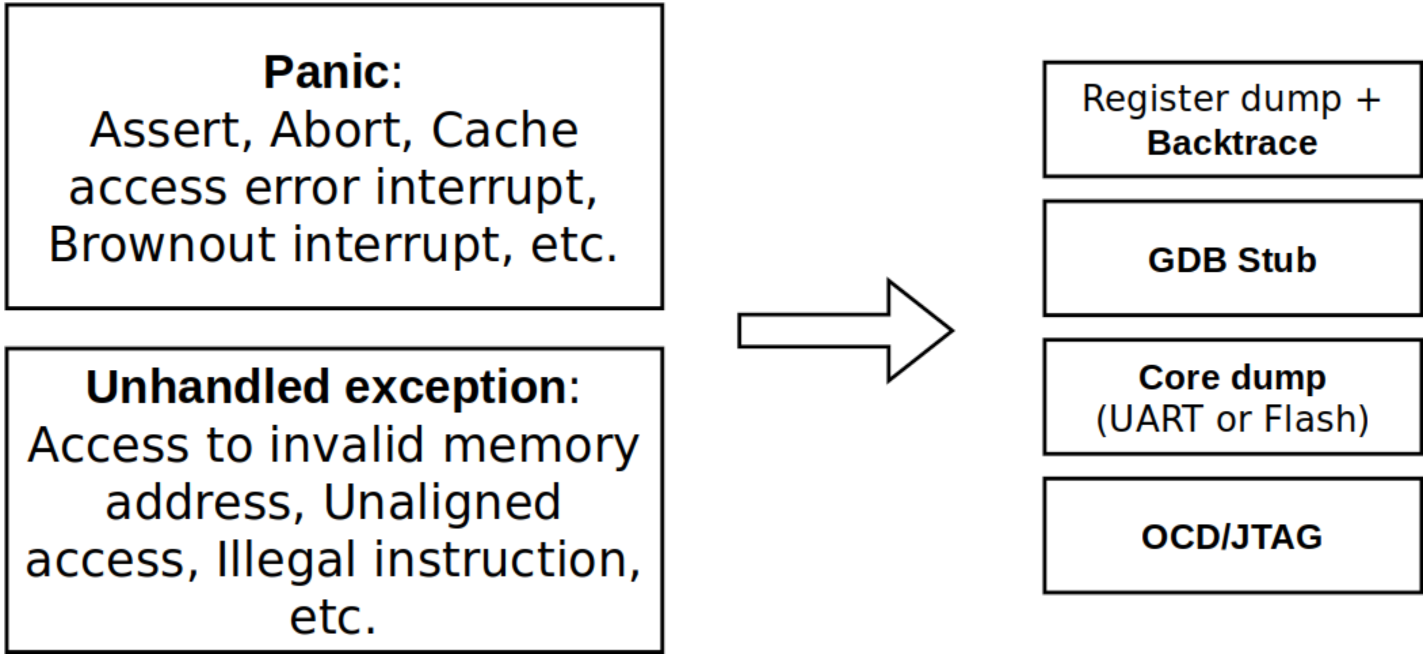
Debugging and Characterizations

Objective

- Debugging
- Application Tracing and Profiling
- Performance Benchmarking
- Memory Footprint Analysis

Debugging

Panic and Exception Handling



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

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

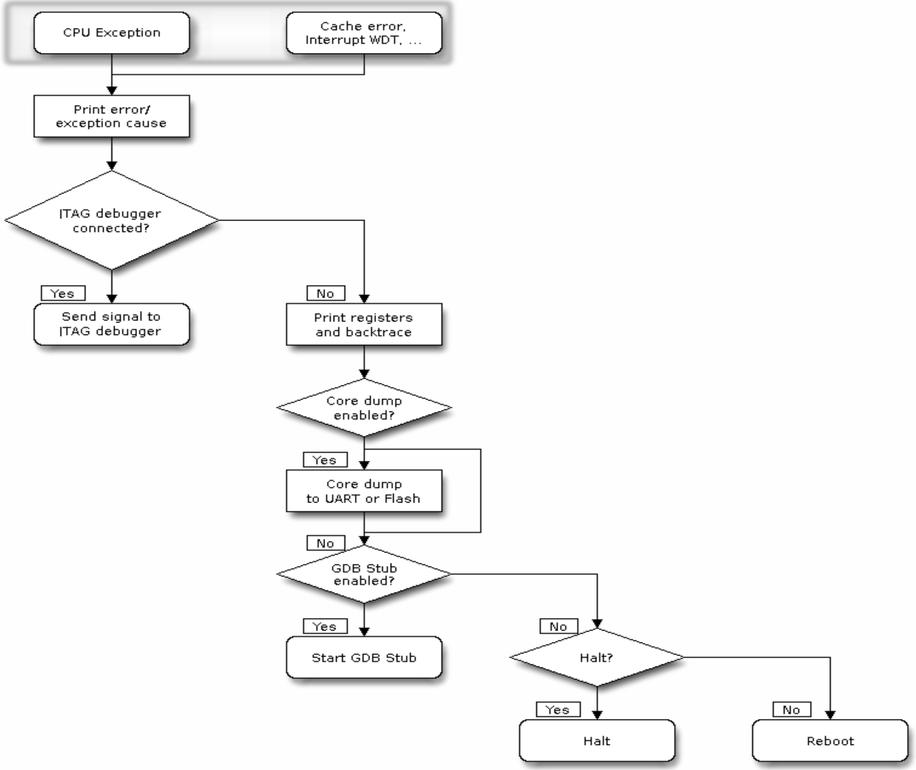
Register dump + **Backtrace**

GDB Stub

Core dump
(UART or Flash)

OCD/JTAG

Panic Handler Workflow



CPU Exception

Cache error, Interrupt WDT, ...

Print error/ exception cause

JTAG debugger connected?

Yes

Send signal to JTAG debugger

No

Print registers and backtrace

Core dump enabled?

Yes

Core dump to UART or Flash

No

GDB Stub enabled?

Yes

Start GDB Stub

No

Halt?

Yes

Halt

No

Reboot

Crash debugging

- Enabled by default and provides following information
  - What kind of exception caused panic
  - On which CPU
  - Where in code (address of program counter)
  - What was the call stack

- IDF Monitor utility helps in decoding backtrace
  - Needs program ELF file to generate backtrace

Crash debugging

```
Guru Meditation Error of type StoreProhibited occurred on core  0. Exception was unhandled.
Register dump:
PC      : 0x400f360d  PS      : 0x00060330  A0      : 0x800dbf56  A1      : 0x3ffb7e00
0x400f360d: do_something_to_crash at /home/gus/esp/32/idf/examples/get-started/hello_world/main/./
hello_world_main.c:57
(inlined by) inner_dont_crash at /home/gus/esp/32/idf/examples/get-started/hello_world/main/./hello_world_main.c:52
A2      : 0x3ffb136c  A3      : 0x00000005  A4      : 0x00000000  A5      : 0x00000000
A6      : 0x00000000  A7      : 0x00000080  A8      : 0x00000000  A9      : 0x3ffb7dd0
A10     : 0x00000003  A11     : 0x00060f23  A12     : 0x00060f20  A13     : 0x3ffba6d0
A14     : 0x00000047  A15     : 0x0000000f  SAR     : 0x00000019  EXCCAUSE: 0x0000001d
EXCVADDR: 0x00000000  LBEG    : 0x4000c46c  LEND    : 0x4000c477  LCOUNT  : 0x00000000

Backtrace: 0x400f360d:0x3ffb7e00 0x400dbf56:0x3ffb7e20 0x400dbf5e:0x3ffb7e40 0x400dbf82:0x3ffb7e60
0x400d071d:0x3ffb7e90
0x400f360d: do_something_to_crash at /home/gus/esp/32/idf/examples/get-started/hello_world/main/./
hello_world_main.c:57
(inlined by) inner_dont_crash at /home/gus/esp/32/idf/examples/get-started/hello_world/main/./hello_world_main.c:52
0x400dbf56: still_dont_crash at /home/gus/esp/32/idf/examples/get-started/hello_world/main/./hello_world_main.c:47
0x400dbf5e: dont_crash at /home/gus/esp/32/idf/examples/get-started/hello_world/main/./hello_world_main.c:42
0x400dbf82: app_main at /home/gus/esp/32/idf/examples/get-started/hello_world/main/./hello_world_main.c:33
0x400d071d: main_task at /home/gus/esp/32/idf/components/esp32/./cpu_start.c:254
```

GDBStub

- Panel Handler Behavior
    - Default behavior is to dump register contents and reset
    - Behavior can be changed to invoke GDBStub

- GDB remote protocol server is started from panic handler

- IDF Monitor utility invokes GDB on host machine once it sees gdb stub is loaded
    - Talks to GDB over UART, can inspect memory and register, and build stack trace

- No special hardware required but not as versatile as JTAG
    - No breakpoints, continue execution or switching between tasks

Coredump

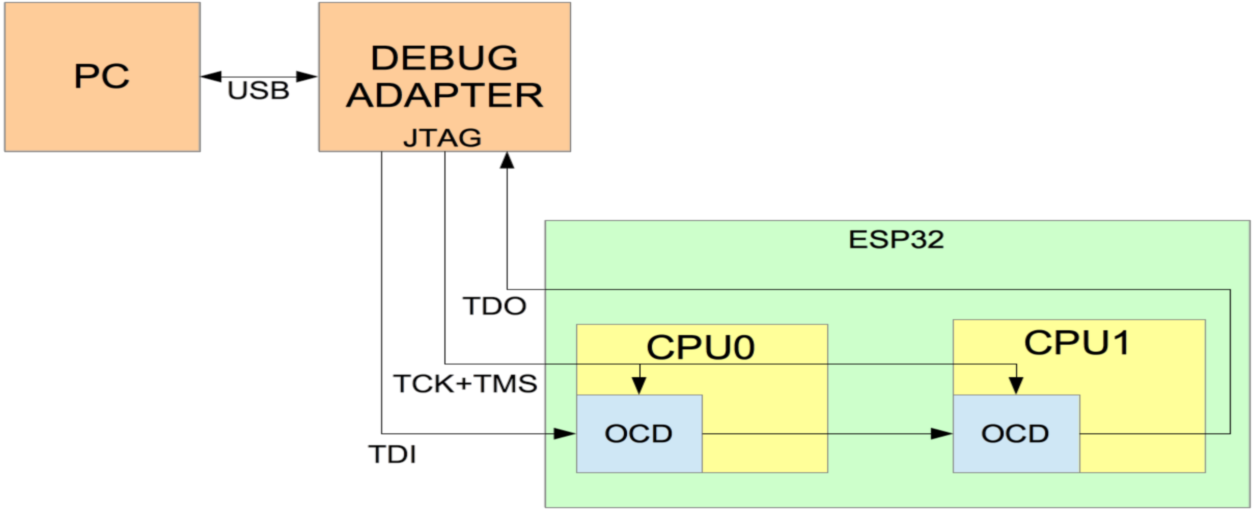
- Core dump module saves stacks and registers of all tasks
  - Configurable option to dump data in Flash partition or on UART

- espcoredump.py tool can decode core dump from Flash or UART

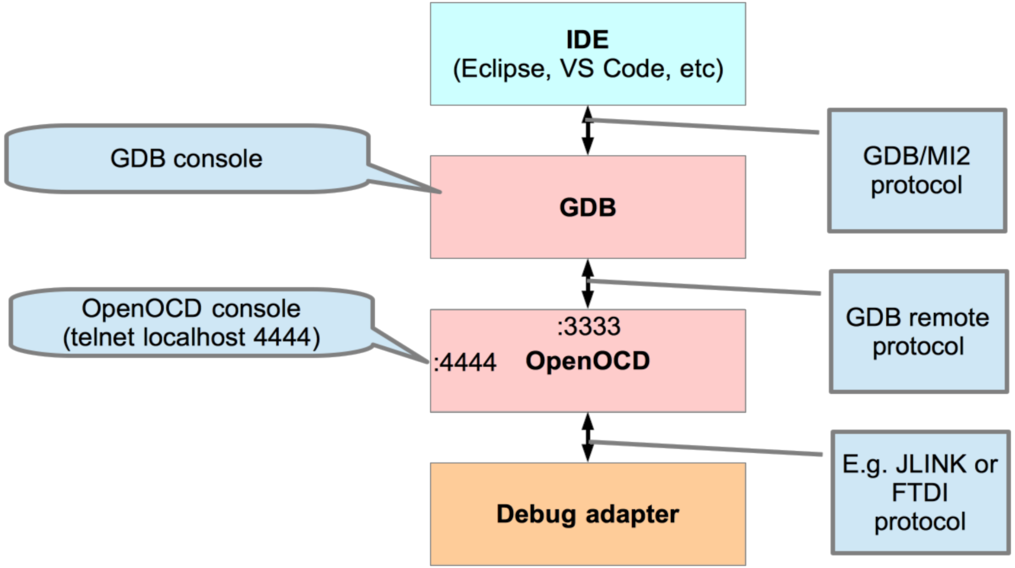
- Launches GDB or prints backtrace

- https://docs.espressif.com/projects/esp-idf/en/v3.1/api-guides/core_dump.html

OpenOCD + JTAG



OpenOCD + JTAG: Software flow



JTAG Debugging Checklist

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

Follow the guide here:
https://docs.espressif.com/projects/esp-idf/en/v3.1/api-guides/jtag-debugging/index.html

Latest version of OpenOCD can be downloaded here:
https://github.com/espressif/openocd-esp32/releases

Stack Overflow Detection

- FreeRTOS provided stack overflow detection mechanism
  - Checks magic bytes placed at the end of stack on each context switch
  - Invokes callback upon finding stack overflow, which looks like below in ESP-IDF

```
void  __attribute__((weak)) vApplicationStackOverflowHook( TaskHandle_t xTask, signed char *pcTaskName )
{
        panicPutStr("***ERROR*** A stack overflow in task ");
        panicPutStr((char *)pcTaskName);
        panicPutStr(" has been detected.\r\n");
        abort();
}
```

Stack Overflow Detection

- Configurable option to enable setting hardware watchpoint at end of stack
  - If enabled on stack overflow will result on unhandled debug exception

- Configurable option to enable stack smashing protection using GCC -fstack-protector* flags

Heap Tracing

- Enables leak checking, memory allocated but never freed

- Heap use analysis - Functions that are allocating/freeing memory while trace is running

```
#include "esp_heap_trace.h"
#define NUM_RECORDS 100
static heap_trace_record_t trace_record[NUM_RECORDS]; // This buffer must be in internal RAM

void app_main()
{
		ESP_ERROR_CHECK( heap_trace_init_standalone(trace_record, NUM_RECORDS) );
}
void some_function()
{
		ESP_ERROR_CHECK( heap_trace_start(HEAP_TRACE_LEAKS) );
		do_something_you_suspect_is_leaking();
		ESP_ERROR_CHECK( heap_trace_stop() );
		heap_trace_dump();
}
```

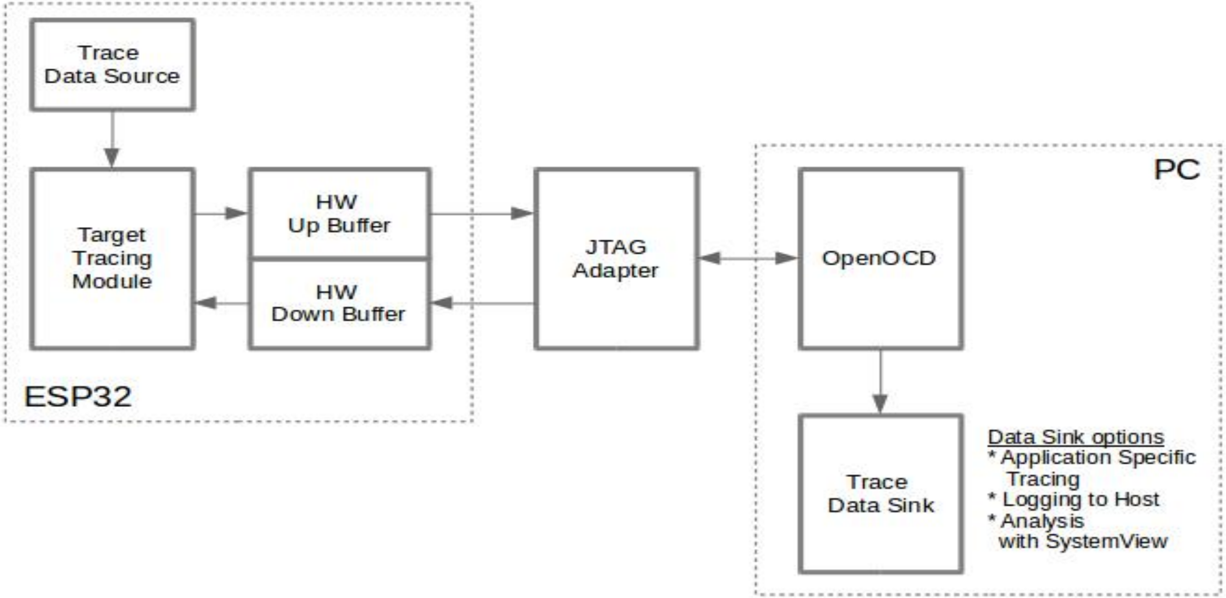
Heap Tracing

```
2 allocations trace (100 entry buffer)
32 bytes (@ 0x3ffaf214) allocated CPU 0 ccount 0x2e9b7384 caller 0x400d276d:0x400d27c1
0x400d276d: leak_some_memory at /path/to/idf/examples/get-started/blink/main/./blink.c:27

0x400d27c1: blink_task at /path/to/idf/examples/get-started/blink/main/./blink.c:52

8 bytes (@ 0x3ffaf804) allocated CPU 0 ccount 0x2e9b79c0 caller 0x400d2776:0x400d27c1
0x400d2776: leak_some_memory at /path/to/idf/examples/get-started/blink/main/./blink.c:29

0x400d27c1: blink_task at /path/to/idf/examples/get-started/blink/main/./blink.c:52

40 bytes 'leaked' in trace (2 allocations)
total allocations 2 total frees 0
```

For more details please refer to,
https://docs.espressif.com/projects/esp-idf/en/v3.1/api-reference/system/heap_debug.html

Application Tracing and Profiling

Application Tracing Module



Data Sink options
* Application Specific Tracing
* Logging to Host
* Analysis with SystemView

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/v3.1/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

Performance Benchmarking

Performance Tuning

- CPU Frequency
  - Default 160 MHz each, Upto 240MHz supported

- Compiler Optimization level
  - Default -Og, for debugging purpose
  - Can be changed to -Os for size optimization

- Flash/SPIRAM Frequency/Mode
  - Default 40 MHz, Dual mode (XIP)
  - Can be changed to 80 MHz, Quad mode

CPU Utilization

- FreeRTOS provides API for collecting run time CPU utilization of various tasks
- Helps to analyze percentage CPU utilization and absolute execution time
- Analysis for ESP32 takes both CPU cores into consideration
- Recommend reading: https://www.freertos.org/rtos-run-time-stats.html

Memory Footprint Analysis

Memory Footprint Analysis

- Static Memory
  - Statically allocated .data and .bss segments in application executable
- Dynamic Memory
  - Available heap memory
  - Application heap utilization
- Flash Footprint
  - Code and .rodata sections from application executable
- Per component Analysis ($IDF_PATH/tools/idf_size.py)