

Firmware Function Reference (TM4C1294 – integr_v03)

This document is a function-by-function reference for the current firmware in this workspace. It focuses on the UART console/diagnostics path and the modules that were actively changed during the recent UART UX + ESP32-feature recovery work.

UART Roles (High-Level)

- **UART3 (USER, 115200)**: interactive console. RX is ISR-driven (USERUARTIntHandler) with echo + in-ISR line editing.
- **UART0 (ICDI, 9600)**: diagnostics/status output. Runtime diagnostics are gated by `DEBUG ON/OFF`.

Session Boundary (DTR on PQ1)

- DTR is read from **PQ1** (DTR_PORT/DTR_PIN) and is **polled in the main loop**.
 - Because DTR is polled (not interrupt-driven), the session loop uses periodic `SysCtlDelay(...)` to ensure disconnects are detected promptly (no “press ENTER to notice disconnect” behavior).
-

main.c

Global state

- `g_ui32SysClock`: system clock (Hz), set by `setup_system_clock()`.
- PWM state:
 - `g_pwmPeriod`: PWM period (ticks).
 - `g_pwmPulse`: PWM pulse width (ticks).
- UART3 RX/command state (ISR-owned, main loop consumes):
 - `user_rx_buf[]`: UART3 line accumulator.
 - `user_rx_len`: current bytes accumulated.
 - `user_cmd_ready`: set by ISR when a non-empty line is completed.
- GOTCHA hidden trigger state (UART3 ISR):
 - `g_uart3_p_run`: count of consecutive P keystrokes.
 - `g_uart3_gotcha_pending`: set when 5 consecutive P are typed.
- UART0 diagnostics gating:
 - `g_debug_enabled`: default false.

`void debug_set_enabled(bool enabled)`

Enables/disables UART0 diagnostic output at runtime.

- Called from the `DEBUG ON/OFF` command via [commands.c](#).

`bool debug_is_enabled(void)`

Returns the current diagnostic gating state.

void pwm_set_percent(uint32_t percent)

Public wrapper used by the command layer.

- Delegates to `set_pwm_percent(percent)`.

void pwm_set_enabled(bool enabled)

Enables/disables PWM output on PF2.

- `enabled=true`: restores PF2 mux to M0PWM2 and enables PWM output.
- `enabled=false`: disables PWM output and reconfigures PF2 as GPIO output low.

This exists primarily to support PSYN OFF for scope/debug, so the tach input can be observed without PWM coupling.

bool pwm_is_enabled(void)

Returns the current PWM output enabled state.

static void set_pwm_percent(uint32_t percent)

Sets PWM duty cycle without disabling/re-enabling the generator.

- Bounds/clamps: percent is clamped to 0..100.
- Ensures pulse width remains in 1..(period-1).
- Calls:
 - `PWMPulseWidthSet(PWM0_BASE, PWM_OUT_2, pulse)`

static void setup_system_clock(void)

Configures system clock to 120 MHz via PLL.

- Sets `g_ui32SysClock`.

static void setup_pwm_pf2(void)

Configures PWM output on PF2 / M0PWM2.

- Enables PWM0 + GPIOF.
- Uses `PWM_SYSCLK_DIV_1`.
- Computes and stores `g_pwmPeriod`.

static void setup_uarts(void)

Configures UART0 and UART3, plus supporting GPIO.

- UART0: PA0/PA1, 9600 8N1.
- UART3: PJ0/PJ1, 115200 8N1.
- PF4 configured as GPIO output (used as RX activity LED and GOTCHA blink).
- PQ1 configured as input with WPU (DTR detect).

- Enables interrupts:
 - INT_UART0 → ICDIUARTIntHandler()
 - INT_UART3 → USERUARTIntHandler()

void ICDIUARTIntHandler(void)

UART0 ISR.

- Echoes RX bytes back to UART0.
- Briefly pulses PNO for visibility.

void USERUARTIntHandler(void)

UART3 ISR: echo + line accumulation + basic line editing.

Behavior summary:

- **Backspace/Delete** (\b or 0x7F):
 - Deletes one buffered character if user_rx_len > 0.
 - Emits "\b \b" erase sequence.
 - If buffer empty, emits bell (\a) to prevent erasing past the prompt boundary.
- **ENTER** (\r or \n):
 - If buffer non-empty: emits \r\n, NUL-terminates buffer, sets user_cmd_ready=true.
 - If buffer empty: does nothing (no extra newline/prompt spam).
- **Uppercase-as-you-type**: converts a..z to A..Z before echo and buffering.
- **Hidden GOTCHA**:
 - Counts consecutive P keystrokes.
 - On 5 consecutive P, sets g_uart3_gotcha_pending=true and resets the counter.
 - Not a command; does not require ENTER; not listed in HELP.
- **Overflow**:
 - Resets buffer and prints ERROR: line too long + prompt.

static void user_uart3_consume_pending_input(void)

Consumes pending UART3 RX bytes while UART3 interrupts are disabled.

Why it exists:

- When hosts send CRLF, the ISR may complete the line on \r and then later receive/echo the trailing \n.
- If the main loop prints the next prompt while UART3 interrupts are disabled, the delayed \n can arrive after the prompt and move the cursor, creating the “extra ENTER required” UX symptom.

What it does:

- While UART3 has bytes available:
 - Swallows extra \r/\n tails.
 - Echoes other bytes and appends them to the current buffer (if a command isn’t already pending).

static void flash_pf4_gotcha(uint32_t flashes)

Flashes PF4 LED flashes times.

- Used when GOTCHA triggers.
- Delay is derived from `g_ui32SysClock` (currently ~75ms on/off).

void example_dynamic_cmd_copy_and_process(const volatile char *user_rx_buf, uint32_t len)

UART0-only diagnostics helper.

- Runs only when `debug_is_enabled()`.
- Uses `diag_uart` helpers to dump internal state and stress some malloc/formatting paths.

int main(void)

Main firmware entry.

Execution overview:

1. Clock + PWM + UART setup.
2. Outer loop waits for DTR session.
3. On session begin:
 - UART0 prints "SESSION WAS INITIATED".
 - UART3 prints rainbow banner + welcome + prompt via `ui_uart3_session_begin()`.
4. Session loop:
 - Polls DTR.
 - If `g_uart3_gotcha_pending` is set:
 - Prints UART0 message immediately.
 - Flashes PF4.
 - If `user_cmd_ready`:
 - Disables UART3 IRQ.
 - Copies buffered line to local storage.
 - Clears ISR-owned state.
 - Calls `user_uart3_consume_pending_input()`.
 - Re-enables UART3 IRQ.
 - Dispatches the command via `commands_process_line(cmd_local)`.
 - If DEBUG enabled: prints additional UART0 diagnostics.
 - Uses a short `SysCtlDelay(...)` to ensure DTR polling stays responsive.
5. On disconnect:
 - UART0 prints "SESSION WAS DISCONNECTED" immediately (no user keystrokes required).

commands.c / commands.h

`commands_process_line()` implements the UART3 command dispatcher.

Design notes:

- Avoids `snprintf/newlib printf`-family in the command response path.
- Uses simple parsing (`strtok_r`) and a small decimal formatter.

void commands_process_line(const char *line)

Parses and executes one complete command line.

- Trims leading whitespace.
- Uppercases command token.
- Supported commands:
 - PSYN *n* — sets PWM duty (5..96).
 - PSYN ON — enables PWM on PF2.
 - PSYN OFF — disables PWM and forces PF2 low.
 - HELP — prints help.
 - DEBUG ON|OFF — gates UART0 diagnostics.
 - EXIT — closes the current UART3 session (no arguments).

void pwm_set_percent(uint32_t percent) (declared in commands.h)

Platform-provided PWM setter (implemented in [main.c](#)).

void debug_set_enabled(bool enabled) / bool debug_is_enabled(void) (declared in commands.h)

Platform-provided debug gating API (implemented in [main.c](#)).

ui_uart3.c / ui_uart3.h

UI helpers for UART3 output discipline (banner/welcome/prompt).

void ui_uart3_session_begin(void)

Prints session-start UI:

- Deterministic ANSI “rainbow banner”.
- A short welcome line.
- A single prompt.

Implementation constraint:

- Session-begin output must be deterministic and not rely on libc-heavy string searching/formatting to avoid stalls.

void ui_uart3_puts(const char *s)

Outputs a C string to UART3 via UARTSend(..., UARTDEV_USER).

void ui_uart3_prompt_once(void)

Prints the prompt once (ANSI_PROMPT + PROMPT_SYMBOL + ANSI_RESET) and avoids duplicate prompt spam.

void ui_uart3_prompt_force_next(void)

Clears the “prompt already printed” latch so the next `ui_uart3_prompt_once()` will print.

diag_uart.c / diag_uart.h

Diagnostics helpers that write to UART0 (ICDI).

Important notes:

- These functions are intended for **non-ISR** contexts.
- The file contains both:
 - heap-based formatting helpers (`diag_vasprintf_heap`, `diag_snprintf_heap_send`) and
 - a minimal printf-like formatter (`diag_simple_sprintf`) plus global `sprintf/snprintf/printf` overrides.

UART0 output primitives

- `diag_putc(char c)`
- `diag_puts(const char *s)`
- `diag_put_hex32(uint32_t v)`
- `diag_put_u32_dec(uint32_t v)`
- `diag_put_ptr(const void *p)`

Heap formatting helpers

- `char *diag_vasprintf_heap(const char *fmt, va_list ap)`
- `char *diag_asprintf_heap(const char *fmt, ...)`
- `int diag_snprintf_heap_send(const char *fmt, ...)`

Memory/allocator diagnostics

- `void diag_sbrk_probe(void)`
- `void diag_test_malloc_with_gpio(void)`
- `void diag_test_malloc_sequence(void)`
- `void diag_print_memory_layout(void)`
- `void diag_print_sbrk_info(void)`
- `void diag_print_variable(const char *name, const void *addr, size_t size, size_t preview_limit)`
- `void diag_print_variables_summary(void)`

Memory protection helpers

- `void diag_check_memory_integrity(const char *context)`
 - `void diag_check_stack_usage(const char *function_name)`
 - `int diag_stack_bytes_used(void)`
 - `int diag_heap_bytes_used(void)`
-

cmdline.c / cmdline.h (legacy)

This module provides an older UART3 command-line loop (`cmdline_run_until_disconnect`) and its own PSYN parsing.

Current status:

- The active firmware path in [main.c](#) uses `USERUARTIntHandler + commands_process_line()`.
- The build includes all `*.c` via the Makefile wildcard; however, link-time garbage collection (`--gc-sections`) typically discards this module unless referenced.

If you decide to use `cmdline_run_until_disconnect()` again:

- It expects a platform-visible `set_pwm_percent(uint32_t)` symbol (currently `set_pwm_percent` is static in `main.c`).
-

tools/uart_session.py (host-side)

Primary host automation tool for UART0/UART3 capture and scripted testing.

Key behaviors:

- Defaults: `--send-delay 0.6, --type-delay 0.02`.
 - Preflight/postflight cleanup is enabled by default; can be disabled via `--no-preflight / --no-postflight`.
 - For testing GOTCHA (real-time keystrokes), prefer `TYPE PPPPP` rather than a line-based `SEND` that appends `ENTER`.
-

Hidden GOTCHA Feature (current spec)

- Trigger: **5 consecutive P keystrokes typed on UART3.**
- Immediate effects:
 - UART0 prints: `GOTCHA: PPPPP detected on UART3.`
 - PF4 LED flashes 5 times.
- Not a command; not listed in `HELP`.