ESP32 Variant Commands & UART-UI Features (Reference)

Artifact name: **ESP32_variant_commands**

Source analyzed: - `otherC/ESP32_SLICKUART_forREFERENCE.c`

This document summarizes the user-visible UART UI/command behavior implemented in that ESP32-oriented reference file, and outlines a non-ISR-invasive plan to integrate similar functionality into the TM4C1294XL firmware variant in this repo.

1) What's in the ESP32 reference file

1.1 UART / session model

- **Single UART console**: Uses UART0 at `115200` (`UART_BAUD`).
- **Polling-based RX**: Main loop polls `UARTCharGetNonBlocking()`.
- **Session activity heuristic (not DTR)**:
  - Maintains a coarse millisecond counter (`g_coarse_ms`) and `g_last_rx_ms`.
  - Declares the session **inactive** after `DISCONNECT_MS` (default `5000ms`) with no received characters.
  - When inactive, the *next received byte* is treated as "reconnect", and triggers a **one-time welcome+prompt**.

1.2 Prompt and output behavior

- **Colored output via ANSI**:
  - `ANSI_WELCOME` (cyan), `ANSI_PROMPT` (yellow), `ANSI_RESPONSE` (green), `ANSI_ERROR` (red), plus `ANSI_RESET`.
- **Prompt deduplication**:
  - Uses a `last_output_was_prompt` flag.
  - `prompt_print_once()` prints the prompt only if it wasn't the most recent output.
- **Welcome message behavior**:
  - On boot: prints exactly once.
  - On reconnect (session transitions inactive → active): prints exactly once.

  Note: this file contains a colored welcome/prompt (cyan/yellow), but it does **not** implement a "rainbow" multi-color banner (unless you consider the ANSI colors as the "rainbow").

1.3 Line editing / input UX

- **Uppercase-as-you-type**:
  - Printable characters are converted to uppercase before echo/storage.
- **Backspace/Delete handling**:
  - `\b` and `0x7F` remove one character if available, echoing `"\b \b"`.
  - If at start of line, emits bell (`\a`).
- **Enter handling**:
  - Treats `\r` and `\n` as end-of-line.
  - Empty line: reprints prompt (only if not already last output).
- **Line length enforcement**:
  - Max line size is `LINEBUF_SIZE` (128). On overflow prints an error and resets line.

1.4 Commands implemented

The reference file implements only one command:

- `PSYN <n>`
  - Parses decimal `n` via `strtol()`.
  - Enforces range `PSYN_MIN..PSYN_MAX` (5..96).
  - On success: sets PWM duty and prints a green `OK: duty set to <n>%` message.
  - On failure: prints a red error message.

  Note: A "GOTCHA" / successive `P` keypress test is **not present** in `ESP32_SLICKUART_forREFERENCE.c` as currently stored in this repo.

2) Key differences vs the TM4C1294XL variant (current repo)

- **TM4C uses DTR on PQ1** as the real session indicator; ESP32 reference uses an inactivity timeout.
- **TM4C UART roles**:
  - UART3: user console (commands)
  - UART0: diagnostics
- **TM4C input path**:
  - UART3 is interrupt-driven (ISR accumulates line buffer and flags `user_cmd_ready`).
  - ESP32 reference is polling-based and owns echo/editing in the polling loop.

3) Non-ISR-invasive integration plan (TM4C1294XL)

Goal: Port the **command repertoire + UI behavior** (welcome/prompt rules, optional editing behavior, GOTCHA-like test if desired) into the TM4C firmware **without changing**: - interrupt schema - low-level UART char handlers / ISR inner mechanics

3.1 Split "command dispatch" from "line acquisition"

Keep the current TM4C approach: - ISR collects bytes into a buffer and raises `user_cmd_ready`.

Refactor at the *top-level* only: - Move command parsing/dispatch into a dedicated module, e.g.: - `commands.c` / `commands.h` - Provide a single entry point: - `commands_process_line(const char *line)`

This preserves the UART ISR design and makes it easy to add new commands.

3.2 Adopt ESP32-style prompt discipline (already mostly solved)

TM4C already has the "extra ENTER" fix applied. To align with ESP32 behavior further (still without ISR changes): - Introduce a `last_output_was_prompt` flag for UART3 output. - Route all UART3 user-visible printing through helpers that clear/set that flag.

3.3 Add colored welcome/prompt (and optionally "rainbow")

Add a `user_session_welcome()` function called from: - the "SESSION WAS INITIATED" path (when DTR asserts)

Implementation notes: - Only prints once per DTR session. - Uses ANSI sequences defined in one header (reuse existing `cmdline.h` ANSI tokens, or define a dedicated `ui_ansi.h`).

If you want a "rainbow" banner: - Implement it as pure output formatting (multiple ANSI color segments) at session start. - This is output-only; it won't affect parsing, ISR behavior, or DTR logic.

3.4 Port "GOTCHA / successive P presses" safely

Since we do not want ISR changes, implement this as a **line-level** feature:

Options: 1) **Command-based**: `GOTCHA` or `PTEST <n>`. - Example: `PTEST 5` prints instructions and then expects the next entered line to be `PPPPP`. 2) **Pattern-based** (still line-level): treat a line consisting of only repeated `P` as the trigger.

This provides the same functional test (detect successive P presses) while staying fully above the UART interrupt layer.

3.5 Preserve existing behavior guarantees

- Keep `PSYN` semantics identical (5..96 range, current PWM update function).
- Keep DTR gating as the authoritative session boundary.
- Keep UART0 diagnostics unaffected (still printed to ICDI).

3.6 Suggested incremental rollout

1) Create `commands_process_line()` with the current `PSYN` implementation.
2) Add `HELP` (prints command list) — low risk.
3) Add "welcome/prompt" polish.
4) Add GOTCHA test as a line-level command.
5) Add any additional commands from the ESP32 lineage.

4) Open items / clarifications

- If you have another ESP32 file/version that contains the GOTCHA code or additional commands, drop it into `otherC/` and I can extract those features too.
- Confirm whether you want the TM4C UART3 console to support **backspace editing** like ESP32 (this typically requires byte-level handling; we can still do it, but it may require careful changes in the UART3 RX ISR, which you asked to avoid for now).