GLOW 2025: Serial Router Firmware Design

State diagram for C++ Firmware on Teensy 4.1

Artur Kraskov
Semester 7
ICT & OI, Delta
Fontys 2025-2026

Contents

Introduction

Within the scope of GLOW 2025 Echoes of Tomorrow serial router firmware was designed. It was built based on the Serial Hardware Communication Agreement received from the team. Prior to the design an analysis of the system was conducted.

This document specifies the design of the Teensy 4.1–based Serial Router as a component of a distributed computer system. It fulfills the requirement to “design a distributed computer system including determining actuators, sensors, timing, resource use and performance” by:

- Defining the Teensy’s role and boundaries within the distributed topology (Mac Mini ⇄ Router ⇄ ESP endpoints/centerpiece).

- Substituting a deeper low-level architectural description (pins, links, signal levels, and protocol framing) for on-device sensors/actuators, which are delegated to ESP endpoints.
- Quantifying timing, resource-use, and performance targets with acceptance hooks and a state-machine model for runtime behavior.

The companion artifacts are the Acceptance/Integration Plan, Technical Advice, and an Analysis & Advice document. The current document extends this prior work [1].

1. System Context (Distributed System)

- **Upstream controller:** Mac Mini via USB CDC (Serial) at 115200 bps.
- **Router:** Teensy 4.1 parses framed commands and routes them to endpoints.
- **Downstream endpoints:** ESP32 devices and a Centerpiece controller via hardware UARTs.
- **Protocol:** `!!<ADDR...>:<TYPE>:<CMD>{<params>}##` per Communication Agreement.

Abstraction of sensors/actuators: Physical sensors and actuators are integrated on the ESP endpoints (LED strips) and Microphones are connected to Mac Mini via a special piece of hardware - these are not covered here. Teensy's responsibility is transport-layer routing, framing enforcement, and confirmations, not direct I/O with sensors/actuators. This document therefore dives deeper into electrical interfaces, buffering, and timing at the router boundary.

2. Interfaces, Pins, and Signals (Teensy 4.1)

- **USB CDC (Serial):** Debug logging and Mac link.
- **UARTs (3.3V TTL):**
  - **Serial1:** Centerpiece (RX1=0, TX1=1) — currently used in reference sketch at 115200 bps.
  - **Serial2:** Arm2 (RX2=7, TX2=8)
  - **Serial3:** Arm3 (RX3=15, TX3=14)
  - **Serial4:** Arm4 (RX4=16, TX4=17)
  - **Serial5:** Arm5 (RX5=21, TX5=20)
  - **Serial6:** Arm1 (RX6=25, TX6=24)

- **Optional GPIO (TBD):**
  - Ready/Busy handshakes per endpoint (active-low with pull-ups recommended).
  - Fault/Watchdog kick lines if required by installation safety.

Electrical and signal integrity constraints:

- **Logic levels:** 3.3V TTL on all Teensy UARTs; use level shifters if any device presents 5V TTL.
- **Cable guidance:** Keep UART runs short with twisted pair and shared ground; shield where needed to reduce EMI.

- **Pin reservation:** Avoid reusing UART pins for interrupts/PWM; document any multiplexing explicitly.

3. Functional Behavior (Derived from Code)

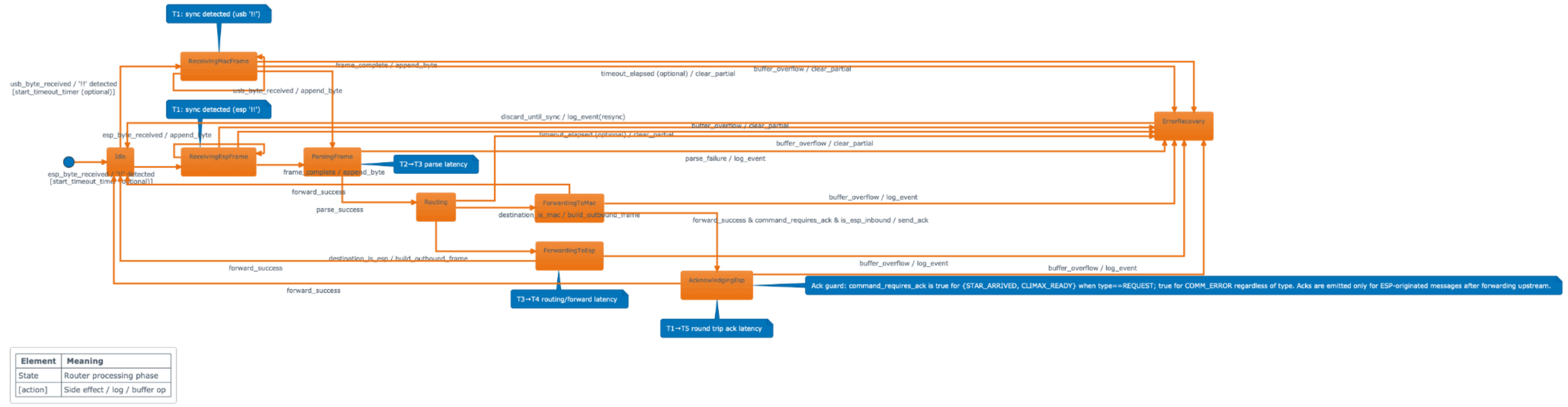
The router ingests two byte streams and performs framed message routing and acknowledgement.

- **Framing:** Detect `!!` prefix and `##` suffix; parse header fields by `:`; optional `{}` parameters.
- **Directionality:** Messages from Mac (`Serial`) must be `REQUEST` and are forwarded to ESP (`Serial1`) with `MASTER` prepended in the address chain. ESP messages are forwarded to the Mac and may trigger acknowledgement to the ESP.
- **Pending correlation:** A small in-memory map associates commands with devices to annotate upstream responses.
- **Logging:** Human-readable logs on `Serial`; optional per-byte tracing for development.

Router state model: See `teensy_router_states.puml` for the orthogonal-line UML state diagram with left-to-right flow starting at Idle. States align with code functions as follows:

- Receiving*Frame ↔ `pumpStream`/`processBuffer`
- ParsingFrame ↔ `parseMessage`
- Routing ↔ `handleMacMessage`/`handleEspMessage`
- ForwardingToEsp ↔ `forwardToEsp`; ForwardingToMac ↔ `forwardToMac`
- AcknowledgingEsp ↔ `sendAckToEsp`

3.1 States Diagram


T1: sync detected (usb '!!')
ReceivingMacFrame
frame_complete / append_byte
usb_byte_received / '!!' detected [start_timeout_timer (optional)]
usb_byte_received / append_byte
timeout_elapsed (optional) / clear_partial
buffer_overflow / clear_partial
T1: sync detected (esp '!!')
discard_until_sync / log_event(resync)
esp_byte_received / append_byte
buffer_overflow / clear_partial
timeout_elapsed (optional) / clear_partial
ErrorRecovery
buffer_overflow / clear_partial
Idle
ReceivingEspFrame
ParsingFrame
T2→T3 parse latency
parse_failure / log_event
esp_byte_received / '!!' detected [start_timeout_timer (optional)]
frame_complete / append_byte
forward_success
Routing
ForwardingToMac
buffer_overflow / log_event
parse_success
destination_is_mac / build_outbound_frame
forward_success & command_requires_ack & is_esp_inbound / send_ack
ForwardingToEsp
destination_is_esp / build_outbound_frame
buffer_overflow / log_event
forward_success
buffer_overflow / log_event
AcknowledgingEsp
forward_success
T3→T4 routing/forward latency
T1→T5 round trip ack latency

Ack guard: command_requires_ack is true for {STAR_ARRIVED, CLIMAX_READY} when type==REQUEST; true for COMM_ERROR regardless of type. Acks are emitted only for ESP-originated messages after forwarding upstream.

Element	Meaning
State	Router processing phase
[action]	Side effect / log / buffer op

Image 1: Serial router States Diagram

High resolution file on GitHub:
https://github.com/GLOW-Delta-2025/serial-router/blob/feature/teensy41_esp32_tx_rx/states_diagram/design/serial_router_diagram.png

4. Timing, Resource Use, and Performance

Baud baselines:

- Upstream USB CDC: 115200 nominal; effective throughput host-dependent.
- Downstream UART: 115200 in reference; Teensy 4.1 supports 230400–921600 with suitable wiring.

Frame time estimates:

- 64-byte frame @115200 ≈ 5.6 ms on the wire (one way). Router overhead adds parsing + logging latency.

Latency budget (target):

- Routing overhead ≤ 2 ms typical without per-byte tracing, ≤ 5 ms worst case under INFO logging.

Buffering and memory:

- `MAX_BUFFER_CHARS = 384` per stream; configurable. Strings incur heap activity; reserve 512–768 B.

Throughput targets:

- Sustained ≥ 100 frames/min across mixed endpoints with no loss and < 20 ms added jitter.

Measurement hooks:

- Timing points: T1 (sync), T2 (frame_complete), T3 (parse_success), T4 (forward_success), T5 (ack_sent) — see diagram notes. Optional microsecond timestamps can be added at these points for empirical validation.
- Metrics to log: frames_routed, frames_malformed, bytes_dropped (overflow), acks_sent.

5. Error Handling and Resilience

- Malformed frames (no `##`) are held until terminator appears; recommend timeout purge (e.g., 250–500 ms).
- Overflow policy: On buffer saturation, drop up to the next `!!` instead of clearing the entire buffer (recommended change).
- Unknown destinations: Forward error/diagnostic upstream; continue normal operation.

Planned improvements (low risk):

- Remove `Serial1.flush()` in `forwardToEsp` to avoid blocking.
- Gate verbose logs on `Serial.dtr()` and add a compile-time `LOG_LEVEL`.
- Implement elapsedMillis timeout per stream to purge stuck partial frames.
- TTL for `PendingEntry` to limit stale correlation.

6. Performance Optimization Options

- Remove `Serial1.flush()`; rely on UART buffering and `availableForWrite()`.
- Pre-reserve `macBuffer`/`espBuffer` and increase `MAX_BUFFER_CHARS` for burst tolerance.
- Replace `String` parsing with fixed ring buffers where long-uptime is critical.
- Add timestamped logs at RX/parse/forward points; disable in production via `LOG_LEVEL`.

Performance acceptance checkpoints:

- Latency: E2E Mac→ESP (64-byte frame) ≤ 10 ms typical at 115200, ≤ 15 ms worst under INFO logging.
- Loss: 0 dropped frames in a 10-minute 100 fpm soak; 0 duplicates.
- Recovery: Parser resynchronizes within 1 frame after malformed input.

7. Security/Safety Considerations

- Electrical: Ensure all links are 3.3V TTL; level-shift any 5V peripherals.
- Protocol: Validate inputs; enforce maximum payload size; avoid buffer overreads.
- Operational: Provide a recovery image and a safe-mode firmware with minimal logging.

8. Open Items / Decisions

- Final baud rates per endpoint (balance between reliability and latency).
- Whether GPIO handshakes are required by any ESP endpoint.
- Memory model hardening (Strings vs. ring buffers) based on soak test results.

9. Traceability

GitHub:
https://github.com/GLOW-Delta-2025/serial-router/tree/feature/teensy41_esp32_tx_rx

- **Acceptance test, integration, technical advice:** `/docs/GLOW 2025 Project Analysis & Advice.pdf`
- **Code Reference:** `/actual_code/final_serial_router_protocol_bridge/teensy_router.ino`
- **State Diagram Plan:** `/states_diagram/design/uml_state_diagram_plan.md`
- **State Diagram:** `/states_diagram/design/teensy_router_states.puml`
- **Design Document:** '/docs/GLOW 2025: Serial Router Firmware Design.pdf'

10. Using the ESP Code to Debug the Teensy Router

The ESP sketch at `actual_code/final_serial_router_protocol_bridge/esp_router.ino` is intended as a bench companion for the Teensy firmware:

- Wire ESP UART to Teensy's selected downstream port (e.g., `Serial1` pins RX1=0, TX1=1 on Teensy 4.1) at the configured baud (115200 by default).
- Use it to send well-formed frames (e.g., `!!ARM1:REQUEST:STAR_ARRIVED{status=ok}##`) and observe Teensy behavior:
- Upstream forwarding to Mac (`forwardToMac`).
- Router confirmations via `sendAckToEsp` for `STAR_ARRIVED`, `CLIMAX_READY`, and `COMM_ERROR`.
- Combine with the state diagram (`teensy_router_states.puml`) to verify transitions Receiving→Parsing→Routing→Forwarding→Acknowledging under realistic traffic.

11. Conclusion

This document provides all necessary details and design that accurately reflects tested code. It was tested with an ESP pair. The code has embedded error handling and is designed to effectively utilize resources of Teensy 4.1. By using it anyone can reproduce the same serial router.

Reference

1. Kraskov A., (2025), GLOW 2025 Project Analysis & Advice. [Online] Available:
   Canvas:
   https://fhict.instructure.com/courses/13084/assignments/271126?module_item_id=1372588
   PDF:
   https://github.com/GLOW-Delta-2025/serial-router/blob/feature/teensy41_esp32_tx_rx/docs/GLOW%202025%20Project%20Analysis%20%26%20Advice.pdf