Hard Real-Time Architectural Invariants

> [!CAUTION] This document defines the absolute, uncompromisable rules of the PAROL6 firmware architecture. If any future implementation (such as STEP/DIR generation, USB transport, or new sensors) violates these invariants, it must be rejected and redesigned.

The entire thesis and academic defensibility of this project rests on the premise of **End-to-End Determinism from ROS to Hardware Capture**. Breaking these rules reverts the system from a "Professional Robotics Stack" back to a "Student Prototype".

Invariant 1: The 1 kHz Golden Tick

The `run_control_loop_isr()` is the sacred heart of the robot. Its performance must never be compromised.

1. **Strictly Bounded Jitter**: The ISR must execute perfectly every 1000 µs. Peak historical jitter is proven to be < 1 µs. If any code addition pushes jitter above 15 µs, the architecture is broken.
2. **Absolute Execution Ceiling**: The entire `sense -> estimate -> supervise -> act` mathematical pipeline must execute in under 25 µs, leaving > 975 µs of CPU headroom for background tasks.
3. **No Dynamic Allocation**: `new`, `malloc`, `String`, or `std::vector::push_back` are strictly forbidden inside the ISR or any class called by the ISR.
4. **No Blocking Operations**: `Serial.print()`, `delay()`, or any `while(!flag)` constructs waiting on external hardware states are strictly forbidden in the ISR.
5. **Cache Coherency**: All arrays, matrices, and state vectors used by the `AlphaBetaFilter`, `Interpolator`, or `ControlLaw` must be placed in Tightly Coupled Memory (TCM) to prevent D-Cache misses from causing execution spikes.

Invariant 2: Hardware-Offloaded Actuation (STEP/DIR)

Creating millions of pulses per second for 6 motors requires massive timing overhead. The CPU must not be involved.

1. **Zero-CPU Generation**: STEP pulses must be generated entirely by hardware IP blocks (e.g., `FlexPWM` or DMA-driven GPIO).
2. **Asynchronous Operation**: The `MotorHAL` output dispatcher inside the 1 kHz ISR is *only* allowed to update hardware registers (e.g., writing a new frequency value). It is forbidden to use `digitalWrite()` or `delayMicroseconds()` to manually bit-bang steps.
3. **Strict Direction Isolation**: Direction changes must incorporate a **1-2 µs hardware dead-time / pause** before resuming STEP pulses to guarantee the motor driver (e.g., MKS Servo42C / TMC5160) registers the change without missing counts.

Invariant 3: Strict Layer Isolation

The mathematical logic must remain entirely ignorant of the physical hardware implementation to maintain simulator compatibility and HIL verifiability.

The Actuation Hierarchy

Do NOT collapse these layers when writing the STEP generation HAL:

1. **`ControlLaw` (Math Layer)**: Outputs ideal `float velocity_command` in pure kinematic `rad/s`. It knows nothing about gears or steps.
2. **`ActuatorModel` (Kinematics Layer)**: Accepts `rad/s`, multiplies by exact non-integer mechanical gear ratios (e.g., 18.0952381), and calculates the required `steps/s`.
3. **`StepScheduler` (Timing Layer)**: Accepts `steps/s` and converts it into the exact register reload values (frequency) required by the hardware timer.
4. **`FlexPWMDriver` (Hardware Layer)**: Blindly applies the frequency registers, handles physical pin muxing, and enforces the 2 µs direction dead-time.

Invariant 4: Simulation & HIL Compatibility

The firmware must be verifiable without physical motors attached.

1. The `MotorHAL` (and underlying `FlexPWMDriver`) must support a `SIMULATION` or `DISABLED` mode where frequency commands are calculated but physical pins are not toggled.
2. The `EncoderHAL` must remain cleanly abstractable. The firmware must boot and execute the 1 kHz ISR correctly regardless of whether it is reading real hardware `QuadTimers` or synthetic spoofed data for CI regression testing.