Part 1

Step 1

The problem is based on an **automated pet feeder** digital control system. The purpose of the system is to dispense dry food at preset times and portions, verify that the intended amount reached the bowl, and raise clear alerts if abnormal conditions occur. The controller (MCU) actuates a **servo-driven gate** to release kibble and uses sensor inputs to make decisions: **a real-time clock** triggers scheduled feeds; a **hopper level sensor** indicates when food is low/empty; and a **bowl load-cell** measures weight before and after dispensing to confirm delivery and, after a wait window, to infer whether the pet has eaten. The logic ensures dispensing only proceeds when hopper level is above a minimum threshold; otherwise, dispensing is inhibited, and a **BIN\_LOW** alert is issued. After actuation, the system checks that the bowl weight increased by approximately the target portion within a tolerance band; if not, it retries a limited number of times and then signals **DISPENSE\_FAIL/JAM**. Following a configurable delay (e.g., 10 minutes), a sufficient weight decrease is treated as evidence of consumption; otherwise, a **NOT\_EATEN** alert is raised. All actions and results (timestamps, target/actual grams, alerts) are logged for traceability. Practical constraints—sensor noise, mechanical variance in servo-to-grams calibration, and limited memory/power—are mitigated by short settle delays, simple filtering (e.g., median), tolerance bands, and safety caps on per-feed and per-day totals. The result is a low-cost, reliable feeder that dispenses accurately, detects faults early, and communicates clearly to the carer.

Step 2

Inputs: the inputs are the sensors/signals

|  |  |
| --- | --- |
| **Name** | **Meaning (binary convention)** |
| **Tfeed** | Indicates current time matches a scheduled feed **not yet served** (1 = feed time, **0** = not feed time ). |
| **BinOK** | Hopper level is **at or above** the minimum threshold (1 = ≥ minimum, **0** = below minimum → low/empty ). |
| **WdeltaOK** | After a dispense attempt, bowl weight increased by at least *(portion − tolerance)* (1 = verified, **0** = not verified ). |
| **RetriesLeft** | There is still at least one retry available (1 = retry allowed, **0** = retry limit reached ). |
| **EatenOK** | After the wait window, bowl weight **decreased** by at least the “eaten” threshold (1 = eaten, **0** = not eaten ). |

**Outputs: the outputs are the control signals**

|  |  |
| --- | --- |
| **Name** | **Meaning (binary convention)** |
| **Vservo** | Actuates the servo to dispense the portion (1 = actuate/open, **0** = idle/closed ). |
| **A\_binlow** | Alert for low/empty hopper (1 = raise alert, **0** = no alert ). |
| **A\_dispfail** | Alert for dispense failure/jam (1 = raise alert, **0** = no alert ). |
| **A\_noteaten** | Alert for food not eaten within window (1 = raise alert, **0** = no alert ). |

**Step 2 — Organize & Describe the Data**

This section defines the inputs, outputs, and key parameters for the automated pet feeder, including realistic sample values and operational constraints.

**Inputs**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Data type / Unit | Source | Purpose in logic | Sample values | Operational constraints / validation |
| RTC time | String HH:MM | Real-time clock | Trigger a feed when time matches schedule | 08:00; 12:30; 18:00 | Polled every 60 s; 24‑hr format; reset served flags at midnight |
| Feeding schedule | List {time, portion\_g} | User configuration | Define feed times & target grams | [(08:00,70), (13:00,40), (18:00,90)] | 1–6 feeds/day; 10–200 g per feed; no overlapping times; integer grams |
| Hopper level | Integer % (0–100) | Level sensor | Blocks dispense when low/empty | 20; 55; 100 | MIN\_BIN\_LEVEL default 15%; debounce ±3%; if ≤ threshold → alert & skip |
| Bowl weight | Float grams (g) | Load cell | Verify dispense & eating | Before 300.0; After 368.5 | Resolution ≈ ±2 g; 3–5 samples after ~5 s settle; median filter; max bowl capacity check |

**Outputs**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Data type | Target/Actuator | When asserted | Sample values | Operational constraints / notes |
| Servo command (dispense) | Action (ms/angle) | Servo gate | On valid feed & hopper OK | ~550 ms ≈ 70 g (calibrated) | Calibrate ms→grams per kibble; up to 2 retries; consider daily cap |
| Alert: BIN\_LOW | Boolean | LED/Buzzer/App | Hopper % ≤ MIN\_BIN\_LEVEL at feed time | 1 (raised) | Latch until refilled/acknowledged; do not dispense |
| Alert: DISPENSE\_FAIL/JAM | Boolean | LED/Buzzer/App | After retries, bowl Δ < (portion − tol) | 1 (raised) | Log last measured Δg; perform maintenance check |
| Alert: NOT\_EATEN | Boolean | LED/Buzzer/App | After wait window, drop < threshold | 1 (raised) | Optional second check 2 min later to reduce false positives |
| Event log | CSV/record | Local storage | Traceability & debugging | 2025‑08‑16 08:00, DISPENSE\_OK, target=70, delta=68 | Compact format; rotate near storage limit; local timestamps |

**Parameters & Thresholds**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Unit | Typical | Allowed range | Why it matters |
| MIN\_BIN\_LEVEL | % | 15 | 5–30 | Prevents running servo on empty hopper |
| DISPENSE\_RETRIES\_MAX | count | 2 | 0–3 | Bounds retry loop; limits mechanical wear |
| PORTION\_TOLERANCE\_G | g | 5 | 2–10 | Absorbs sensor noise & pellet bounce |
| EAT\_WINDOW\_MIN | min | 10 | 5–60 | Time allowed before eaten check |
| EAT\_MIN\_DROP\_G | g | 10 | 5–30 | Minimum bowl-weight drop to infer eating |
| POLL\_INTERVAL\_SEC | s | 60 | 30–120 | Main loop cadence; minute‑aligned |
| MAX\_DAILY\_G (optional) | g/day | 300 | 150–600 | Safety cap to avoid over‑feeding |

**Notes**

G = gram

Min = minute

S = second

g/day = grams per day

Step 2 defines what the system reads, does, and tunes so the algorithm is testable and safe. Inputs are **RTC time, feeding schedule {time, portion\_g}, hopper level (%)**, and **bowl weight (g)** for dispense/eating verification. Outputs are the **servo command, alerts** (BIN\_LOW, DISPENSE\_FAIL, NOT\_EATEN), and **event logs.** Key parameters set constraints: **MIN\_BIN\_LEVEL≈15%, DISPENSE\_RETRIES\_MAX=2, PORTION\_TOLERANCE\_G≈5 g, EAT\_WINDOW\_MIN≈10 min, EAT\_MIN\_DROP\_G≈10 g.** These definitions enable consistent calibration, simulation, and edge-case handling.

**Step 3**

**Step 3 — Plan the Solution (Algorithm & Flowchart Spec)**

This section states the feeder’s control algorithm, Boolean rules, truth tables, and a flowchart specification suitable for Draw.io/diagrams.net.

**A) Algorithm (high-level)**

1) Init: Load SCHEDULE and constants (MIN\_BIN\_LEVEL, DISPENSE\_RETRIES\_MAX, PORTION\_TOLERANCE\_G, EAT\_WINDOW\_MIN, EAT\_MIN\_DROP\_G).

2) Main loop (every 60 s): If current time equals a not-yet-served schedule item, proceed; else loop.

3) Bin check: If hopper% ≤ MIN\_BIN\_LEVEL → raise BIN\_LOW alert, log, and skip dispense.

4) Pre-dispense: Read pre = bowl weight (g). Set retries = 0; verified = false.

5) Dispense & verify (retry up to DISPENSE\_RETRIES\_MAX): actuate servo, wait ~5 s, compute delta = bowl\_now − pre; if delta ≥ (portion\_g − tolerance) → verified=true else retries++.

6) If not verified after retries → raise DISPENSE\_FAIL/JAM alert and loop.

7) If verified → wait EAT\_WINDOW\_MIN; compute drop = pre + portion\_g − bowl\_now.

8) If drop ≥ EAT\_MIN\_DROP\_G → log PET\_ATE; else raise NOT\_EATEN alert.

9) Mark the feed as served and continue looping; reset served flags at midnight.

**B) Boolean control rules (signals)**

Signals: Tfeed, BinOK, WdeltaOK, RetriesLeft, EatenOK; Outputs: Vservo, A\_binlow, A\_dispfail, A\_noteaten. (1=true, 0=false)

Vservo = Tfeed ∧ BinOK

A\_binlow = Tfeed ∧ ¬BinOK

Retry = ¬WdeltaOK ∧ RetriesLeft

Vservo = Vservo ∨ Retry # actuate again if Retry=1

A\_dispfail = (Tfeed ∧ BinOK) ∧ ¬WdeltaOK ∧ ¬RetriesLeft

A\_noteaten = Verified ∧ ¬EatenOK # Verified latches high when any attempt yields WdeltaOK=1

**C) Truth tables**

1) Initial dispense / bin check

|  |  |  |  |
| --- | --- | --- | --- |
| Tfeed | BinOK | Vservo | A\_binlow |
| 0 | X | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

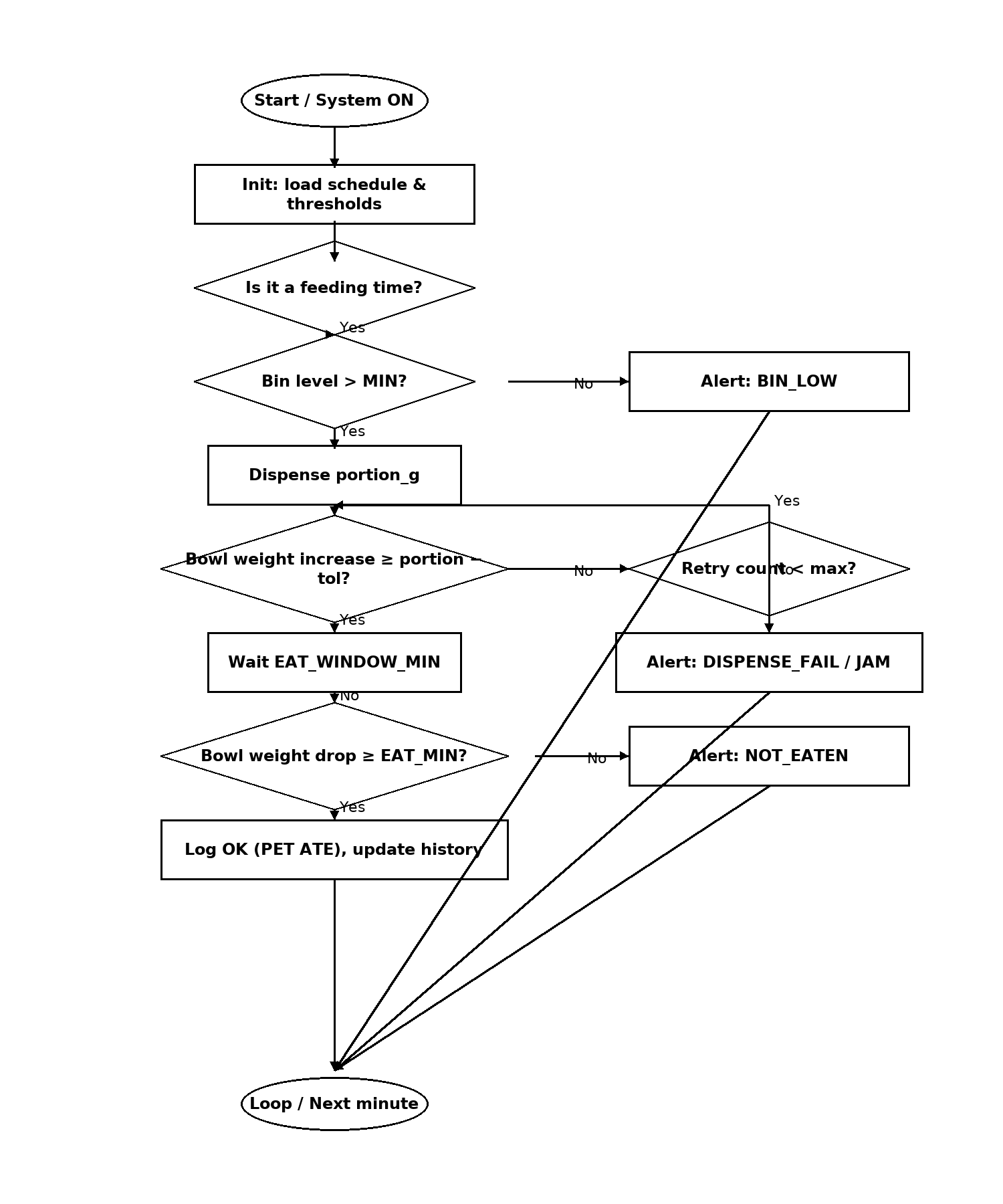
2) Post-attempt verify / retry / fail

|  |  |  |  |
| --- | --- | --- | --- |
| WdeltaOK | RetriesLeft | Next Vservo (retry) | A\_dispfail |
| 1 | X | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 |

3) Eaten check (after wait window)

|  |  |
| --- | --- |
| EatenOK | A\_noteaten |
| 1 | 0 |
| 0 | 1 |

**D) Flowchart specification (for Draw.io)**

****

**Step 4 — Implement the Solution (Word Code / Plain-English Pseudocode)**

**CONSTANTS / SETTINGS**

MIN\_BIN\_LEVEL = 15%

DISPENSE\_RETRIES\_MAX = 2

PORTION\_TOLERANCE\_G = 5 g

EAT\_WINDOW\_MIN = 10 minutes

EAT\_MIN\_DROP\_G = 10 g

POLL\_INTERVAL\_SEC = 60

SCHEDULE = [{time: HH:MM, portion\_g: int }, …]

**SENSORS / ACTUATORS (abstracted)**

s\_rtc\_time () -> 'HH:MM'

s\_bin\_pct () -> 0.100 (percent)

s\_bowl\_g () -> float grams (filtered, after ~5 s settle)

a\_servo\_dispense(portion\_g) # actuate servo to drop grams

a\_alert(kind, message) # LED/buzzer/app notification

a\_log (event, data) # compact CSV/flash log

**STATE**

served\_today = {time: bool} # all false at start/midnight

daily\_total\_g = 0 # optional safety cap

**INIT**

1) Load SCHEDULE and constants.

2) Set all served\_today[t] = false.

3) Read baseline bowl weight if needed.

**MAIN LOOP (every POLL\_INTERVAL\_SEC)**

4) now = s\_rtc\_time()

5) If no schedule at 'now' OR served\_today[now] == true: continue

6) Bin check:

if s\_bin\_pct() <= MIN\_BIN\_LEVEL:

a\_alert('BIN\_LOW', 'Hopper near empty');

a\_log('BIN\_LOW', { time: now });

continue

7) pre = s\_bowl\_g(); retries = 0; verified = false

**DISPENSE & VERIFY**

8) do:

a\_servo\_dispense(portion\_g\_at(now))

wait ~5 s (pellets settle)

delta = s\_bowl\_g() - pre

if delta >= portion\_g\_at(now) - PORTION\_TOLERANCE\_G:

verified = true

else:

retries += 1

9) while (verified == false) and (retries <= DISPENSE\_RETRIES\_MAX) repeat step 8

10) if verified == false:

a\_alert('DISPENSE\_FAIL', 'Portion not detected; possible jam')

a\_log('DISPENSE\_FAIL', { time: now, delta: delta })

continue

**POST-FEED EATEN CHECK**

11) wait EAT\_WINDOW\_MIN

12) drop = pre + portion\_g\_at(now) - s\_bowl\_g()

13) if drop >= EAT\_MIN\_DROP\_G:

a\_log('PET\_ATE', { time: now, portion: portion\_g\_at(now), delta: delta, drop: drop })

else:

a\_alert('NOT\_EATEN', 'Food not eaten within window')

a\_log('NOT\_EATEN', { time: now, portion: portion\_g\_at(now), drop: drop })

**FINALISE THIS FEED**

14) served\_today[now] = true

15) daily\_total\_g += portion\_g\_at(now) # optional daily cap

16) At midnight: reset served\_today and daily\_total\_g

**Part 5 — Test & Refine (Debug & Verify)**

This section lists core test cases with expected outcomes, a quick test procedure, refinements if tests fail, and final acceptance criteria.

**A) Test cases (core matrix)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Scenario | Preconditions (Given) | Steps | Expected results |
| T1 | Eats as expected | Schedule 08:00→70g; Bin 60%; Bowl 300g | At 08:00 run loop → dispense once (settle ~5s) | Bowl ↑ ≥ 65g (portion−tol); wait 10 min; bowl drop ≥ 10g → Log: PET\_ATE; no alerts |
| T2 | Not eaten | Schedule 18:00→90g; Bin 55%; bowl stable | Dispense verified → wait 10 min | Bowl drop < 10g → Alert: NOT\_EATEN; log NOT\_EATEN |
| T3 | Bin low/empty | Schedule 08:00→70g; Bin 5% | Loop reaches 08:00 | Skip dispense → Alert: BIN\_LOW; log BIN\_LOW |
| T4 | Jam / under-dispense | Schedule 08:00→70g; Bin 40% | Attempt 1: bowl Δ small; attempt 2: bowl Δ small; attempts = max (2) | Not verified after retries → Alert: DISPENSE\_FAIL; no eaten check |
| T5 | Partial then success | Schedule 08:00→70g; Bin 50% | Attempt 1: Δ=30g; Attempt 2: cumulative Δ ≥ 65g | Verify success → wait 10 min → apply eaten check (drop ≥ 10g passes else NOT\_EATEN) |
| T6 | Over-portion guard | Daily cap 300 g; today served 260 g | Next feed 70 g due | Clamp/block to respect cap; log clamp or Alert: DAILY\_CAP (if implemented) |
| T7 | No scheduled feed | No time match; normal sensors | Run loop for 5 min | No actuation; no alerts; idle logging only |
| T8 | Sensor noise near tolerance | Portion 70 g; tol 5 g; noisy readings ±2 g | Dispense once; apply 5 s settle + median filter | Verified if median Δ ≥ 65g; otherwise, a single retry occurs |

Download the CSV version if needed for spreadsheets: Step5\_Test\_Cases.csv

**B) How to run the tests (quick procedure)**

• Set constants: MIN\_BIN\_LEVEL=15%, DISPENSE\_RETRIES\_MAX=2, PORTION\_TOLERANCE\_G=5 g, EAT\_WINDOW\_MIN=10 min, EAT\_MIN\_DROP\_G=10 g.

• For each test, set schedule/bin/bowl state and advance the clock to the feed time.

• Observe: bowl deltas, retry count, alerts, and log entries.

• Record measured Δ (g) after dispensing and drop (g) after the wait window.

• Pass/Fail = expected alert/log exactly matches the table.

**C) Refinements if a test fails**

• False under-dispense increase settle to 7–8 s, apply median of 3–5 samples, or widen tolerance to ±6–7 g.

• Too many retries reduce DISPENSE\_RETRIES\_MAX to 1; recalibrate ms→grams mapping.

• False NOT\_EATEN (slow eater): add a second check 2–3 min later or reduce EAT\_MIN\_DROP\_G to ~8 g.

• BIN\_LOW oscillation: add hysteresis (raise at 15%, clear at 18%).

• Over-feeding risk: enforce MAX\_DAILY\_G and log clamping.

• Noisy bowl readings: median filter + discard outliers > 3σ; verify load-cell zero/stability.

• Midnight rollover issues: reset 'served today' flags at 00:00; store date with each served entry.

**D) Acceptance criteria (final)**

For each scheduled feed, the system must either: (a) dispense within tolerance and correctly classify PET\_ATE vs NOT\_EATEN, or (b) emit the correct alert (BIN\_LOW / DISPENSE\_FAIL) without unsafe behavior (no endless retries, no over-portion).