**Step 4: Implement the Solution**

* **Purpose**
  + Design the step-by-step control logic for a low-cost automated pet feeder that dispenses food on schedule, verifies dispensing, checks if the pet ate, and alerts staff to problems.
* **Definitions (inputs, outputs, parameters)**

**Inputs / Sensors / Settings**

* + current\_time - from Real-Time Clock (format HH:MM)
  + feeding\_schedule - list of times, e.g., [08:00, 18:00]
  + portion\_size\_g - grams per meal, e.g., 120
  + food\_level\_state - one of {FULL, OK, LOW, EMPTY}
  + bowl\_weight\_g - weight measured by bowl sensor (grams)
  + (optional) door\_open, network\_ok, hopper\_weight\_g

**Outputs / Actuators**

* + servo\_dispense(cycles) - rotates the servo to drop food
  + buzzer(pattern), status\_led(pattern)
  + send\_alert(reason, details)
  + log\_event(record)

**Operational Parameters**

* + dispense\_cycle\_g = 30 (grams per sevo cycle)
  + max\_dispense\_cycles = 6
  + consumption\_wait\_min = 10
  + consumed\_delta\_g = max(15, 0.20\*portion\_size\_g) (grams)
  + low\_food\_threshold\_% = 20
* **High-Level Sequence (top-level loop)**

**TASK A - System Start & Load**

1. Power on System
2. Load feeding\_schedule and portion\_size\_g
3. Calibrate sensors (tare bowl, verify readings)
4. Set status\_led = GREEN steady
5. log\_event(“SYSTEM\_START”)

**TASK B - Continuous Monitoring Loop**

1. Repeat forever:

6.1 Read current\_time

6.2 If current\_time matches any time in feeding\_schedule and this slot not yet served today → go to TASK C (Scheduled Feed).

6.3 Otherwise, remain idle/monitor (brief sleep, then loop).

* **Scheduled Feed (detailed tasks)**

**TASK C - Pre-checks before dispensing**

1. Read food\_level\_state
2. If food\_level\_state == EMPTY →
   1. send\_alert(“HOPPER\_EMPTY”, “Refill required”)
   2. Status\_led = RED blink, buzzer = short\_beep\_3
   3. log\_event(“FEED\_SKIPPED\_EMPTY”)
   4. Return to monitoring loop
3. (Optional Safety) If door\_open == true →
   1. send\_alert(“DOOR\_OPEN”, “Close lid/door”)
   2. log\_event(“FEED\_BLOCKED\_DOOR”)
   3. Return to monitoring loop
4. Read and store bowl\_weight\_before = bowl\_weight\_g
5. Compute needed\_cycles = ceil(portion\_size\_g / dispense\_cycle\_g); cap at max\_dispense\_cycles.

**TASK D - Dispense portion**

1. For i from 1 to needed\_cycles:

6.1 Command servo\_dispense(1 cycle); wait for cycle to complete

6.2 (Optional jam check) If servo reports overload or timeout →

* send\_alert(“DISPENSE\_FAILURE”, “Possible jam at cycle “ + i)
* Status\_led = RED blink, buzzer = short\_beep\_3
* log\_event(“DISPENSE\_FAIL”, {cycle:i})
* Break and return to monitoring loop

1. After final cycle, read bow\_weight\_after\_dispense = bowl\_weight\_g
2. If (bowl\_weight\_after\_dispense - bowl\_weight\_before) < (0.5 \* dispense\_cycle\_g) (i.e., almost no increase) →

* send\_alert(“DISPENSE\_FAILURE”, “No measurable food drop”)
* log\_event(“DISPENSE\_NO\_INCREASE”)
* Return to monitoring loop

1. log\_event(“DISPENSE\_SUCCESS”, {portion\_g: portion\_size\_g, cycles: needed\_cycles})

**TASK E - Wait and verify consumption**

1. Wait consumption\_wait\_min minutes
2. Read bowl\_weight\_after\_wait = bowl\_weight\_g
3. Compute consumed\_g = (bowl\_weight\_after\_dispense - bowl\_weight\_after\_wait)
4. If consumed\_g >= conusmed\_delta\_g →

* Mark feed “EATEN”
* Status\_led = GREEN steady
* log\_event(“CONSUMED\_OK”, {consumed\_g})
* Return to monitoring loop

1. Else →

* send\_alert(“NOT\_EATEN”, “Pet did not eat within wait window”)
* status\_led = AMBER blink, buzzer = soft\_beep\_1
* log\_event(“NOT\_CONSUMED”, {consumed\_g})
* Return to monitoring loop
* **Manual Feed (optional but recommended)**

**TASK F - Manual Override**

1. If manual\_feed\_button pressed:

1.1 Validate food\_level\_state != EMPTY and door\_oprn == false

1.2 Dispense manual\_poriton\_g usign the same TASK D logic

1.3 Set a short lockout(e.g., 3 minutes) to avoid repeated presses

1.4 log\_event(“MANUAL\_FEED”)

* **Alerts (standard reasons and actions)**
  + HOPPER\_EMPTY - prompt refill; LED red; buzzer pattern; optional SMS/app
  + DISPENSE\_FAILURE - possible jam or empty; check mechanism and hopper
  + NOT\_EATEN - pet skipped meal; staff should inspect pet/bowl
  + DOOR\_OPEN - close lid/door to continue

Each alert calls:

* send\_alert(reason, details)
* status\_led pattern update
* Buzzer pattern (if allowed)
* Log\_event with timestamp and context
* **Data Logging (reocmmended minimum fields)**

For each significant event:

timestamp, event\_type, schedule\_time, portion\_g, cycles, bowl\_weight\_before, bowl\_weight\_after\_dispense, bowl\_weight\_after\_wait, consumed\_g, food\_level\_state, result, alerts[].

* **Explanations (how each task maps to goals)**
  + TASK A/B ensure continuous operation and trigger only at configured times.
  + TASK C prevents unsafe/pointless dispensing (empty hopper, door open).
  + TASK D performs controled dispensing and detects jams or non-delivery.
  + TASK E verifies pet consumption using a simple, measurable threshold.
  + TASK F supports humane/manual intervention with safety checks.
  + Alerts and Logging create traceabiility for staff action and future imporovements.

**Short version solution**

START

Load schedule and portion; calibrate sensors; LED=GREEN; LOG start.

LOOP:

Read current\_time.

IF current\_time is a scheduled slot not yet served THEN

IF food\_level\_state == EMPTY THEN ALERT "HOPPER\_EMPTY"; LOG; CONTINUE.

IF door\_open == TRUE THEN ALERT "DOOR\_OPEN"; LOG; CONTINUE.

Read bowl\_weight\_before.

Compute needed\_cycles = ceil(portion\_size\_g / dispense\_cycle\_g), cap at max.

DISPENSE needed\_cycles with jam checks; IF jam THEN ALERT "DISPENSE\_FAILURE"; LOG; CONTINUE.

Read bowl\_weight\_after\_dispense.

IF (after\_dispense - before) < minimal\_increase THEN ALERT "DISPENSE\_FAILURE"; LOG; CONTINUE.

Wait consumption\_wait\_min.

Read bowl\_weight\_after\_wait; consumed\_g = (after\_dispense - after\_wait).

IF consumed\_g >= consumed\_delta\_g THEN LOG "CONSUMED\_OK"; LED=GREEN.

ELSE ALERT "NOT\_EATEN"; LOG "NOT\_CONSUMED".

ENDIF

Sleep briefly and repeat.

END LOOP