Weekend Assignment

1. Smart Home Temperature Control Problem Statement:
   Design a temperature control system for a smart home. The system should read the current temperature from a sensor every minute and compare it to a user-defined setpoint.

   Requirements:
   - If the current temperature is above the setpoint, activate the cooling system.
   - If the current temperature is below the setpoint, activate the heating system.
   - Display the current temperature and setpoint on an LCD screen.
   - Include error handling for sensor failures.

   Pseudo code:

   Initialize temperature sensor, LCD display.

   Define a setpoint.

   Read temperature every one minute
   current_temperature = read_temp_sensor();

   If (current_temperature == INVALID)
   Display "ERROR" on LCD
   Skip remaining steps and go to the next loop.
   Else
   Display "CURRENT TEMPERATURE" and "SETPOINT" on LCD

   If (current_temperature > setpoint)
   Activate cooling.
   Deactivate heating.
   Else if (current_temperature < setpoint)
   Activate heating.
   Deactivate cooling.
   Else (when current_temerature == setpoint)
   Deactivate both heating and cooling

   Delay for 1 minute

Flowchart


Start
Read temperature
False
Current temprature == INVALID
True
Display current temperature and set point
Display ERROR
False
current_tem perature > setpoint
True
Delay for 1 min
Activate cooling & deactivate heating
False
current_tem perature < set_point
True
Deactivate heating & cooling
Activate heating & deactivate cooling

2.Automated Plant Watering System Problem Statement:
Create an automated watering system for plants that checks soil moisture levels and waters the plants accordingly.

Requirements:
• Read soil moisture level from a sensor every hour.
• If moisture level is below a defined threshold, activate the water pump for a specified duration.
• Log the watering events with timestamps to an SD card.
• Provide feedback through an LED indicator (e.g., LED ON when watering).

Pseudo code :

Initialize soil moisture sensor, water pump, SD card, LED indicator.

Set moisture_threshold.

Set pump_duration.

soil_moisture = read_moisture_sensor();

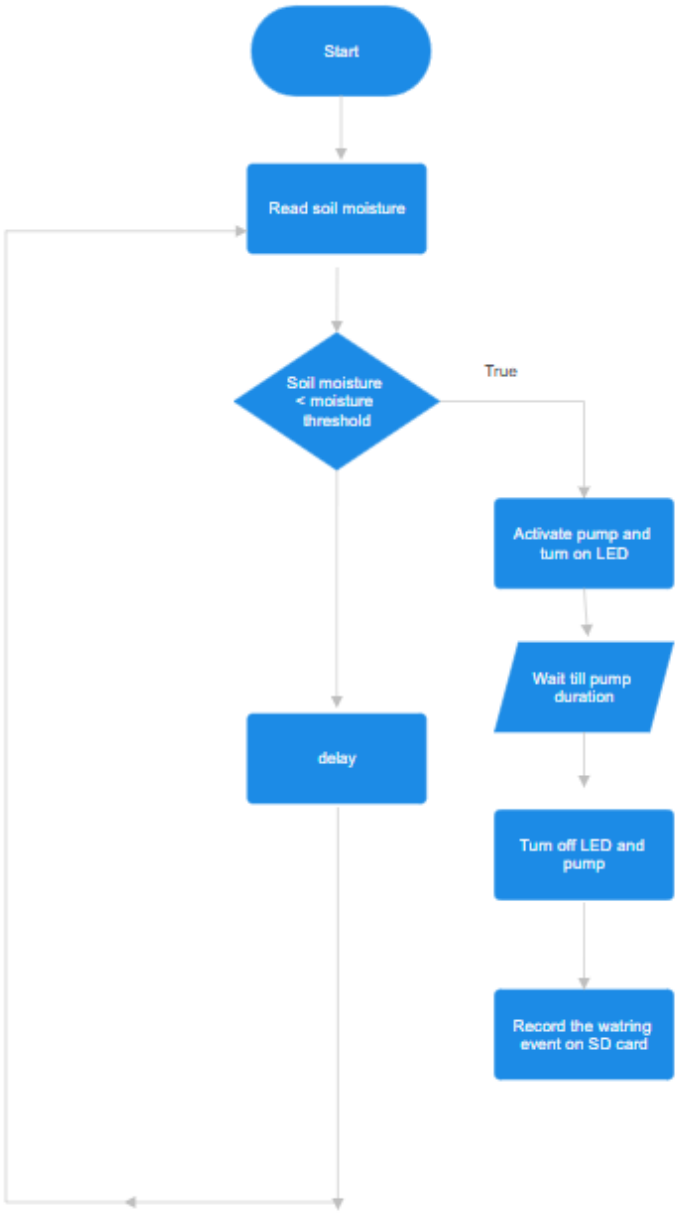
If soil_moisture < moisture_threshold
Activate watering
- Turn on water pump.
- Turn on LED indicator.
- Keep the pump running for specified duration (delay of pump_duration).
- Turn off water pump.
- Turn off LED indicator.
- Record the watering event with timestamp on SD card.

Else

No action required

Delay for 1 hour.

Flowchart:


Start
Read soil moisture
Soil moisture < moisture threshold
True
Activate pump and turn on LED
Wait till pump duration
Turn off LED and pump
Record the watring event on SD card
delay

3. Motion Detection Alarm System Problem Statement:
Develop a security alarm system that detects motion using a PIR sensor.

Requirements:
• Continuously monitor motion detection status.
• If motion is detected for more than 5 seconds, trigger an alarm (buzzer).
• Send a notification to a mobile device via UART communication.
• Include a reset mechanism to deactivate the alarm.

Pseudo code :

Initialize PIR sensor, buzzer, UART communication, reset button.
Initialize a flag motion_detected = 0; // indicates motion is not detected
moisture_duration_threshold = 5 seconds.

motion_detected = read_PIR_sensor();

IF(motion_detected)

- Record the start time.
- Continuously check the motion lasts 5 seconds.

IF motion_duration >= motion_duartion_threshold

- Turn on buzzer
- Send notification to mobile via UART communication

Else

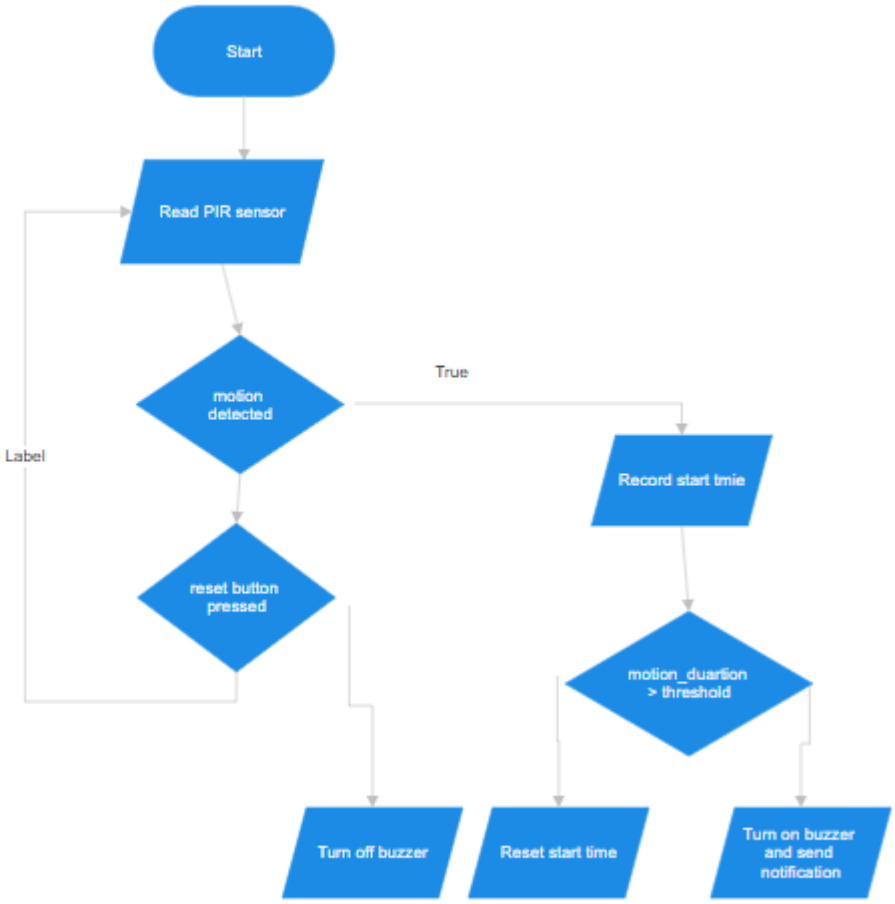
- Reset start time

IF reset_button == PRESSED

- Turn off buzzer

Delay for continuous monitoring .

Flowchart


Start
Read PIR sensor
motion detected
True
Label
Record start tmie
reset button pressed
motion_duartion > threshold
Turn off buzzer
Reset start time
Turn on buzzer and send notification

4. Heart Rate Monitor Problem Statement:
   Implement a heart rate monitoring application that reads data from a heart rate sensor.

   Requirements:
   • Sample heart rate data every second and calculate the average heart rate over one minute.
   • If the heart rate exceeds 100 beats per minute, trigger an alert (buzzer).
   • Display current heart rate and average heart rate on an LCD screen.
   • Log heart rate data to an SD card for later analysis.

   Pseudo code :

   Initialize heart rate sensor, LCD display, buzzer, SD card.

   Set high_heart_rate_threshold = 100 BPM.

   Set sampling_interval = 1 second.

   Declare current_heart_rate, heart_rate_sum, sample_count .
   Initialize sample_count =0;

   Main loop

   current_heart_rate = read_heart_rate_sensor()

   heart_rate_sum += current_heart_rate .
   ++sample_count.

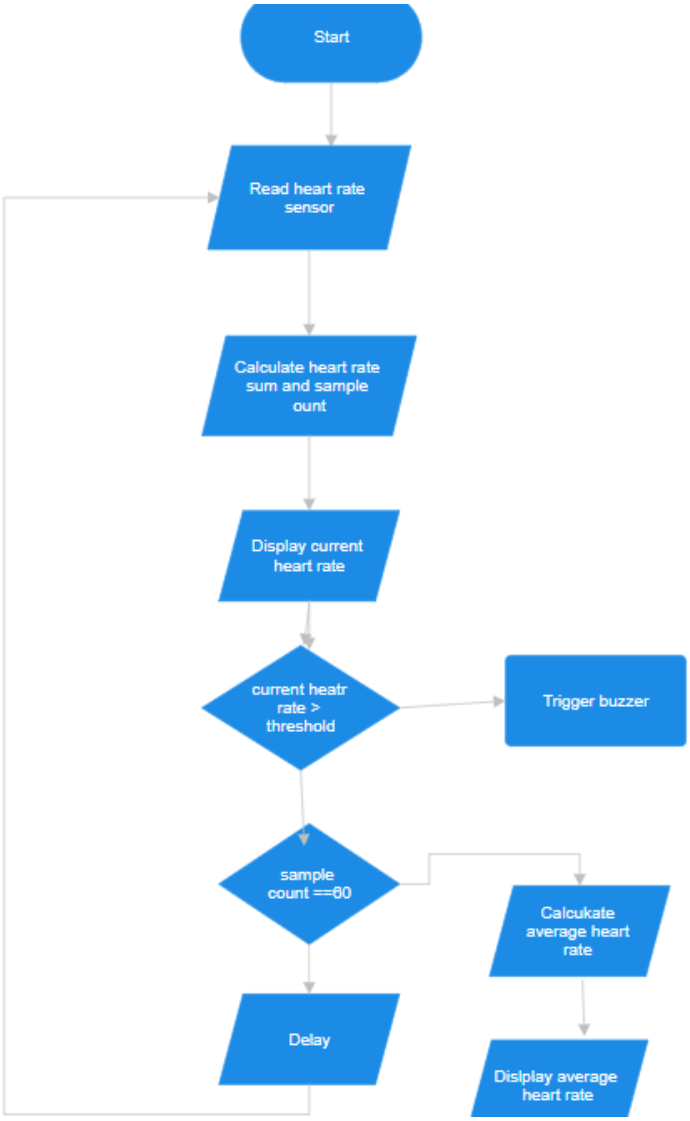
   Display current_heart_rate on LCD

   If current_heart_rate > high_heart_rate_threshold
     Trigger the buzzer

   If sample_count == 60
   - average_heart_rate = heart_rate_sum / sample_count.
   - Display average_heart_sum on LCD
   - Log the average_heart_sum and sample_count to SD card
   - Reset heart_rate_sum and same_count to 0.

   Delay for 1 second.

Flowchart


Start
Read heart rate sensor
Calculate heart rate sum and sample ount
Display current heart rate
current heatr rate > threshold
Trigger buzzer
sample count ==60
Calcukate average heart rate
Delay
Dislplay average heart rate

5. LED Control Based on Light Sensor Problem Statement:
Create an embedded application that controls an LED based on ambient light levels detected by a light sensor.
Requirements:
• Read light intensity from the sensor every minute.
• If light intensity is below a certain threshold, turn ON the LED; otherwise, turn it OFF.
• Include a manual override switch that allows users to control the LED regardless of sensor input.
• Provide status feedback through another LED (e.g., blinking when in manual mode).

Pseudo code:

Initialize light sensor, control LED, status LED.

Set light_threshold

Initialize the flag to indicate manual_mode = 0 (inactive).
If override switch is pressed.
If(manual_mode)
- Blink LED to indicate manual_mode is active

Else

- Turn off LED.

If(manual_mode == 0)

light_intensity = read_light_sensor()

If light_intensity < light_threshold:

Turn ON the control LED.

Else:

Turn OFF the control LED.

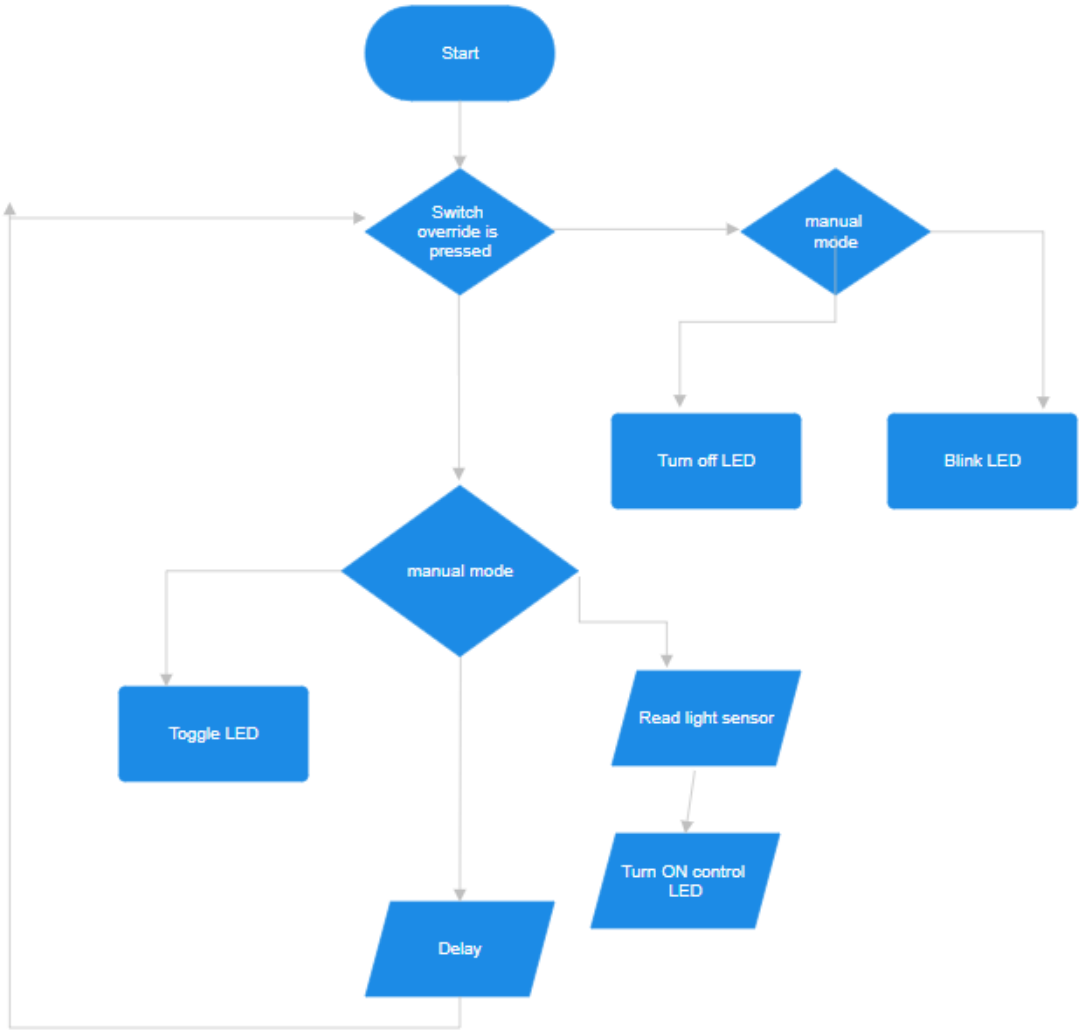
Else

If switch is pressed

- Toggel state of LED

Delay for 1 minute.

Flowchart


Start
Switch override is pressed
manual mode
Turn off LED
Blink LED
manual mode
Toggle LED
Read light sensor
Turn ON control LED
Delay

6. Digital Stopwatch Problem Statement:
Design a digital stopwatch application that can start, stop, and reset using button inputs.

Requirements:
• Use buttons for Start, Stop, and Reset functionalities.
• Display elapsed time on an LCD screen in hours, minutes, and seconds format.
• Include functionality to pause and resume timing without resetting.
• Log start and stop times to an SD card when stopped.

Pseudo code:

Initialize LCD display, buttons ,SD card.
Declare
- is_running = False. //track stop watch is running
- is_paused = False. //track stop watch is paused
- start_time //Store starting time stamp
- elapsed_time = 0 //Store elapsed time stamp

Main loop
If Start Button == PRESSED

If is_running == False:
- Set is_running = True.
- Set is_paused = False.
- Set start_time to the current timestamp.
- Reset elapsed_time to 0.

Else if is_running == True and is_paused == True (resume from pause):

- Set is_paused = False.
- Set start_time = current time - elapsed_time.

Else if Stop Button == PRESSED
If is_running == True and is_paused == False:
- Set is_running = False.
- Calculate and log start_time and elapsed_time to the SD card.
- Display final time on LCD.
- Set elapsed_time = 0

Else if Reset button == PRESSED

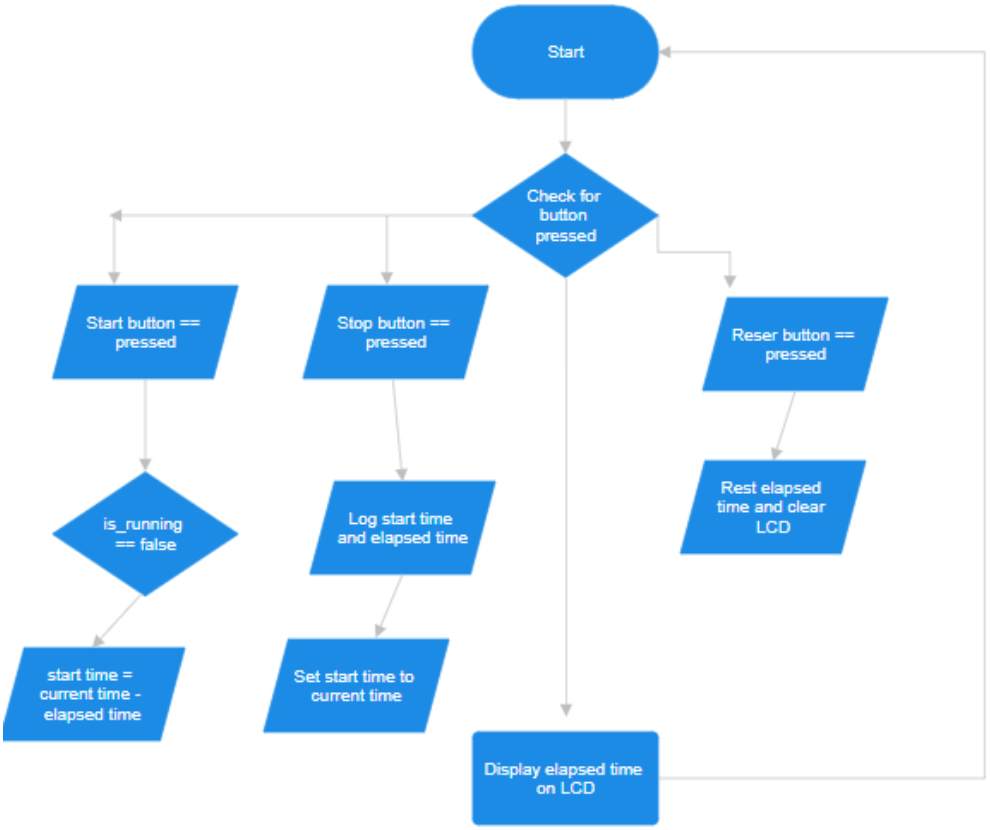
- Set is_running = False.
- Set is_paused = False.
- Reset elapsed_time to 0.
- Clear display on LCD.

If is_running == True and is_paused == False:

- elapsed_time = current_time - start_time
- Call display_time(elapsed_time) to update the time on LCD

Delay for checking button press and update display.

Flow chart



7.Temperature Logging System Problem Statement:
Implement a temperature logging system that records temperature data at regular intervals.

Requirements:
• Read temperature from a sensor every 10 minutes.
• Store each reading along with its timestamp in an array or log file.
• Provide functionality to retrieve and display historical data upon request.
• Include error handling for sensor read failures.

Pseudo code :

Initialize temperature sensor, display.

Set logging_interval = 10 minutes.
Declare

- temperature_log = [] //an array for storing timestamped temperature data
- current_temperature // to store the latest temperature reading.
- current_timestamp //to store the time of each reading.

current_temperature = read_temp_sensor()

current_timestamp = get_current_timestamp()

If current_temperature == INVALID (sensor read failure):

- Log “ERROR” with current_timestamp in the temperature_log.

Else

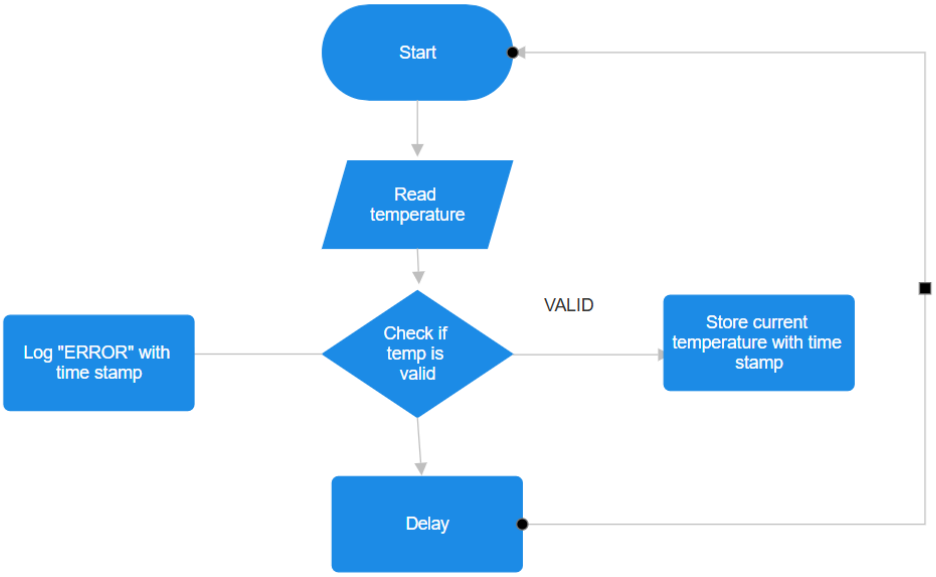
- Store current_temperature and current_timestamp in temperature_log.

Delay for logging_interval

If user requests historical data:

- Access and display entries from temperature_log on the display (or console).

Flow chart


Start
Read temperature
Check if temp is valid
VALID
Log "ERROR" with time stamp
Store current temperature with time stamp
Delay

7. Bluetooth Controlled Robot Problem Statement:
Create an embedded application for controlling a robot via Bluetooth commands.

Requirements:
• Establish Bluetooth communication with a mobile device.
• Implement commands for moving forward, backward, left, and right.
• Include speed control functionality based on received commands.
• Provide feedback through LEDs indicating the current state (e.g., moving or stopped).

Pseudo code:

Initialize Bluetooth module, motor control pins, speed control, LEDs.
Define Commands

- "F" for moving forward.
- "B" for moving backward.
- "L" for turning left.
- "R" for turning right.
- "S" for stopping.
- "1" to "5" for speed levels (1 = slowest, 5 = fastest).

Define

- current_speed = 0
- current_state = "stopped"

Main loop

command = read_bluetooth()

If command == "F":

- Set motor control pins for forward movement.
- Update current_state = "moving forward".

Else if command == "B":

- Set motor control pins for backward movement.
- Update current_state = "moving backward".

Else if command == "L":

- Set motor control pins for turning left.
- Update current_state = "turning left".

Else if command == "R":

- Set motor control pins for turning right.
- Update current_state = "turning right".

Else if command == "S":

- Stop all motors.
- Update current_state = "stopped"

If commad is a speed command (“1” to “5”)

- Convert command to integer speed_level.
- Set current_speed = speed_level.
- Adjust motor speed using PWM based on current_speed.

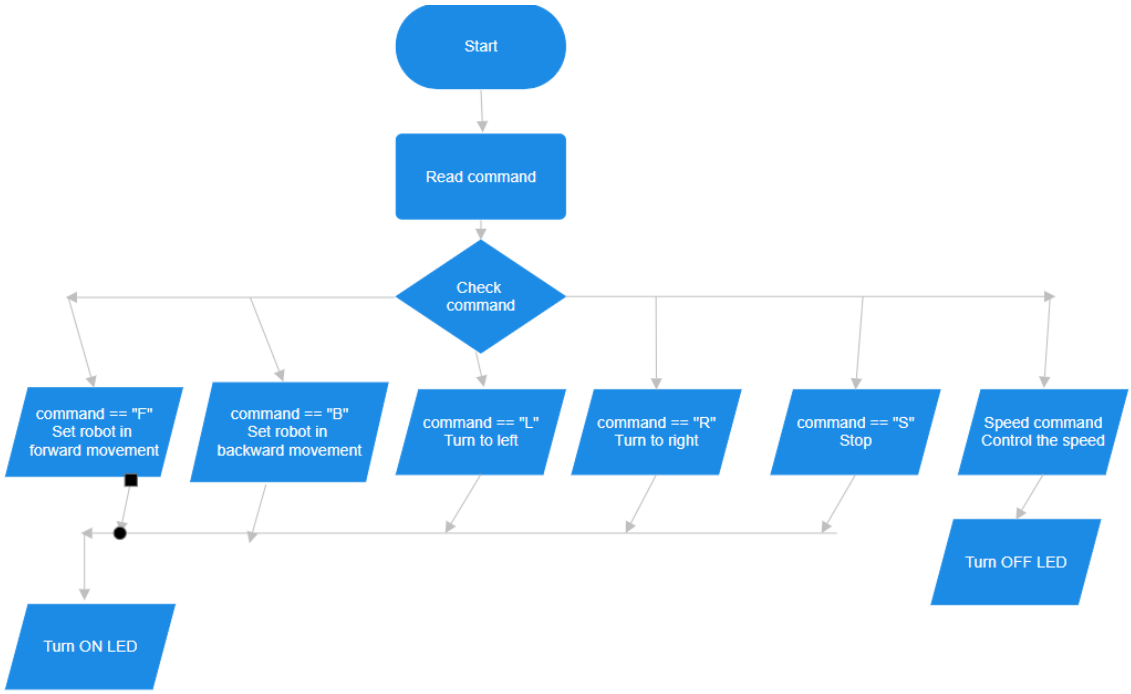
If current_state == "stopped":

- Turn off all state LEDs.

Else:

- Turn on an LED to indicate the robot is moving
  Delay for continuous command processing.

Flowchart



9.Battery Monitoring System Problem Statement:

Develop a battery monitoring system that checks battery voltage levels periodically and alerts if voltage drops below a safe threshold.

Requirements:

- Measure battery voltage every minute using an ADC (Analog-to-Digital Converter).
- If voltage falls below 11V, trigger an alert (buzzer) and log the event to memory.
- Display current voltage on an LCD screen continuously.
- Implement power-saving features to reduce energy consumption during idle periods.

Pseudo code:

Initialize ADC, LCD, buzzer.

Set voltage_threshold = 11.0 volts .

Set measurement_interval = 1 minute.

is_alert_triggered = False // to track whether an alert has been triggered.

current_voltage = read_adc_voltage()

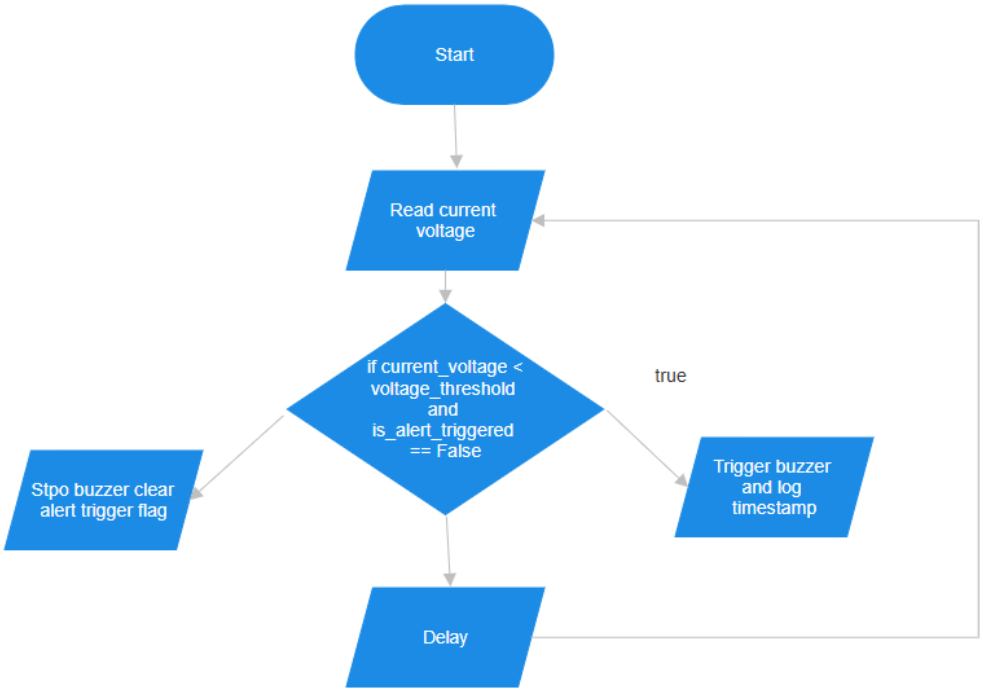
if current_voltage < voltage_threshold and is_alert_triggered == False:

- Trigger the buzzer for alert.
- Log the timestamp and current_voltage to memory.
- Set is_alert_triggered = True to avoid repeated alerts.

Else if current_voltage >= voltage_threshold:

- Stop the buzzer.
- Set is_alert_triggered = False
- Enable power-saving mode (e.g., sleep mode) to reduce power consumption.
- Delay for measurement_interval (1 minute)

Flowchart



10.RFID-Based Access Control System Problem Statement:

Design an access control system using RFID technology to grant or deny access based on scanned RFID tags.

Requirements:

- Continuously monitor for RFID tag scans using an RFID reader.
- Compare scanned tags against an authorized list stored in memory.
- Grant access by activating a relay if the tag is authorized; otherwise, deny access with an alert (buzzer).
- Log access attempts (successful and unsuccessful) with timestamps to an SD card.

Pseudo code:

Initialize RFID reader, relay, buzzer, SD card, load authorized tag list.
scanned_tag = read_rfid_tag()

timestamp = get_current_timestamp()

If scanned_tag is in authorized_tags:

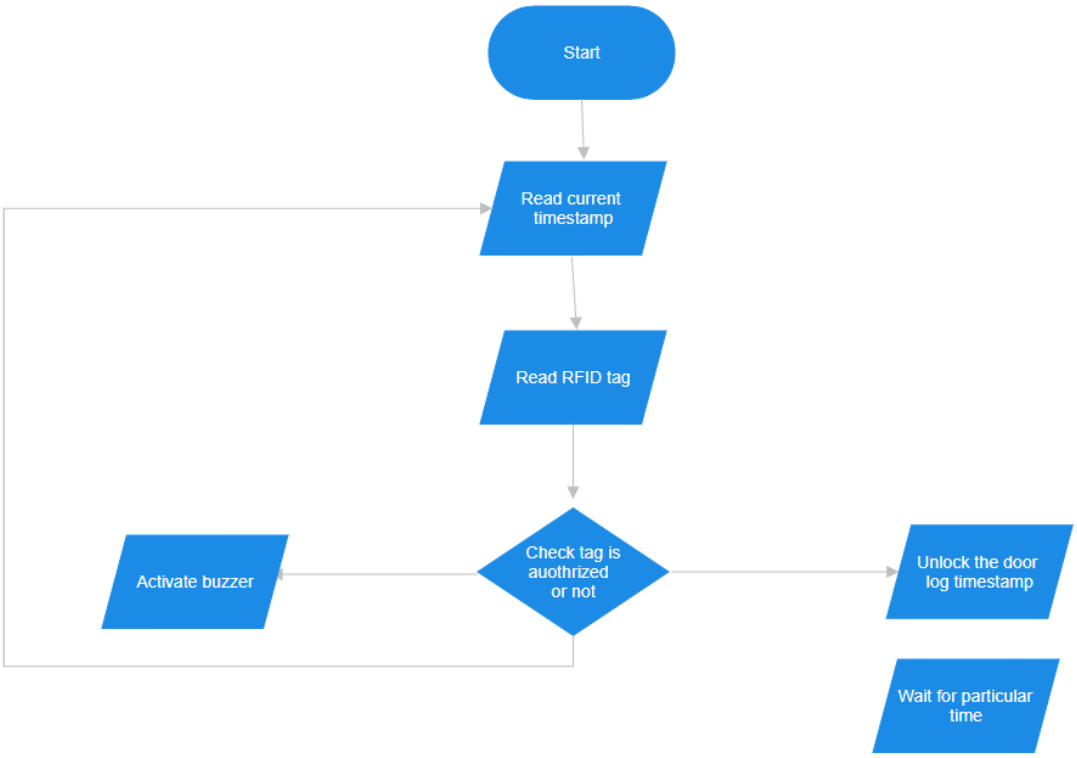
- Activate relay to unlock door.
- Log "Access Granted" with scanned_tag and timestamp to SD card.
- Keep relay activated for a few seconds (e.g., 5 seconds) to allow entry.
- Deactivate relay to re-lock door.

Else:

- Activate buzzer to signal access denial.
- Log "Access Denied" with scanned_tag and timestamp to SD card.
- Keep buzzer active for a short time (e.g., 2 seconds) then turn it off

Delay of 100ms se

Flowchart


Start
Read current timestamp
Read RFID tag
Check tag is auothrized or not
Activate buzzer
Unlock the door log timestamp
Wait for particular time

x