Aleida Diaz-Roque

Professor Elkaim

ECE 118

5 April 2024

Check-off: 04/05/2025

Lab 0 - The Roach

PreLab

Part 1 - PCB Assembly and Soldering

- What temperature should you set the iron to?

  The temperature should range between 600˚-700˚F (315˚-370˚C) but it will depend on the type of solder used. For Tin-Lead solder, the temperature should be on the lower end, about 330˚C and for Silver solder about 350˚C.

- What are the differences in how to handle Tin-Lead vs. Silver solder?

  Tin-Lead has a lower melting point than Silver and it will flow more easily. In general, because lead is toxic, one must be very careful after handling it. Washing hands is important regardless of which solder is used but even more essential for Tin-Lead

- How can you tell a hot weld from a cold weld?

  A hot weld references a shiny, smooth weld while a cold one is dull and may refer to an improperly joined connection implying that the solder did not melt completely.

- What does black in the weld mean?

  Black can indicate oxidation or a problem with the flux such as charring if the temperature was too high. They could also indicate some kind of contaminant in the weld.

Part 4 - Roach Hardware Exploration

```
# Test Harness for Roach
# many variables will be #defines to control

# Spin the specified motor in the given direction and speed
def test_motor(motor, direction, speed):
     if direction == BACKWARD:
          speed = -speed

     if motor == RIGHT:
          Roach_RightMtrSpeed(speed)
     elif motor == LEFT:
          Roach_LeftMtrSpeed(speed)
     # Display speed on LED bar
     display_speed_on_LED(speed) # speed/100 * 12
     # use: Roach_BarGraph(uint8_t Number)
     delay(SHORT)

# Display bumper status individually or collectively
def read_and_display_bumpers(individual):
    if individual:
          if bumper_input == FRONT_LEFT:
               front_left_status = Roach_ReadFrontLeftBumper()
               back_left_status = Roach_ReadBackLeftBumper()
               print("Front Left Bumper:", front_left_status)
               print("Back Left Bumper:", back_left_status)

          elif bumper_input == FRONT_RIGHT:
               front_right_status = Roach_ReadFrontRightBumper()
               back_right_status = Roach_ReadBackRightBumper()
               print("Front Right Bumper:", front_right_status)
               print("Back Right Bumper:", back_right_status)

          # Display current battery voltage for a duration
          display_battery_voltage(30)  # Read for 30s

     else:
          # Read and print collective bumper states
          bumpers_status = Roach_ReadBumpers()
          print("All Bumpers: {bumpers_status:04b}")
          # Display as 4-bit binary

# Display current battery voltage for the specified duration
```

```
def display_battery_voltage(duration):
     start_time = current_time()
     while current_time() - start_time < duration:
          battery_voltage = Roach_BatteryVoltage()
          print("Battery Voltage:", battery_voltage)
          delay(SHORT)  # Prevent flooding


# Read/display light sensor value for the specified duration
def display_light_level(duration):
     start_time = current_time()
     while current_time() - start_time < duration:
          light_sensor_value = Roach_LightLevel()
          print("Light Sensor:", light_sensor_value)
          delay(SHORT) # Prevent flooding

main():

     while (1):
          bumper_input = Roach_ReadBumpers()

          if bumper_input == FRONT_LEFT:
               test_motor(LEFT, FORWARD, FAST)
               read_and_display_bumpers(IND) # individual
               display_battery_voltage(30)

          elif bumper_input == FRONT_RIGHT:
               test_motor(RIGHT, FORWARD, FAST)
               read_and_display_bumpers(IND)
               display_battery_voltage(30)

          elif bumper_input == REAR_LEFT:
               test_motor(LEFT, BACKWARD, SLOW)
               read_and_display_bumpers(CUML)
               display_light_level(30)  # Read for 30s

          elif bumper_input == REAR_RIGHT:
               test_motor(RIGHT, BACKWARD, SLOW)
               read_and_display_bumpers(CUML)
               display_light_level(30)  # Read for 30s

          else:
               print("Invalid input")
```

The test harness for the Roach involves conducting various tests controlled by bumper inputs to assess the functionality of the roach's components. Pressing the front left or right bumpers initiates tests for the respective side's motor, running it forward at a high speed, and also displays the status of both the front and back bumpers on that side. During these tests, the current battery voltage is monitored and displayed for 30 seconds.

On the other hand, engaging the rear left or right bumpers triggers the corresponding motor to operate backward at a slow speed. In these cases, the collective status of all bumpers is shown, and the light sensor's value is continuously displayed for 30 seconds. Motor speeds are visually represented on an LED bar graph, offering a direct, real-time gauge of operational status. This comprehensive testing suite ensures that each component of the roach—motors, bumpers, battery voltage, and light sensors—can be individually assessed and verified for proper functionality.

Part 5 - Event Detection

```
# BUMPER EVENT CHECKER

define: BUMPER_NOT_TRIPPED 0
define: BUMPER_TRIPPED 1
static uint8_t lastBumperState = BUMPER_NOT_TRIPPED

function CheckBumperEvent() returns uint8_t:
    currentBumperState = Roach_ReadFrontLeftBumper()
                         # changed for all bumpers

    if currentBumperState == BUMPER_TRIPPED and lastBumperState
== BUMPER_NOT_TRIPPED:
        PostEvent(BUMPER_EVENT, BUMPER_TRIPPED)
        lastBumperState = BUMPER_TRIPPED
        return TRUE
    else:
        lastBumperState = currentBumperState
        return FALSE
```

```
# LIGHT LEVEL EVENT CHECKER

define: DARK_THRESHOLD
define: LIGHT_THRESHOLD
static enum {DARK, LIGHT} lastLightState = LIGHT

function CheckLightLevel() returns uint8_t:
    currentLightValue = Roach_LightLevel()
    currentLightState = lastLightState

    if currentLightValue > DARK_THRESHOLD:
        currentLightState = DARK
    elif currentLightValue < LIGHT_THRESHOLD:
        currentLightState = LIGHT

    if currentLightState != lastLightState:
        PostEvent(LIGHTLEVEL_CHANGE, currentLightState)
        lastLightState = currentLightState
        return TRUE
    else:
        return FALSE
```

```
# BATTERY CHECK EVENT

define: BATTERY_LOW_THRESHOLD SomeValue
static enum {BATTERY_OK, BATTERY_LOW} lastBatteryState =
BATTERY_OK

function CheckBatteryEvent() returns uint8_t:
    currentBatteryLevel = Roach_BatteryVoltage()
    currentBatteryState = lastBatteryState

    if currentBatteryLevel < BATTERY_LOW_THRESHOLD:
        currentBatteryState = BATTERY_LOW
    else:
        currentBatteryState = BATTERY_OK

    if currentBatteryState != lastBatteryState:
        PostEvent(BATTERY_STATUS_CHANGE, currentBatteryState)
```

```
        lastBatteryState = currentBatteryState
        return TRUE
    else:
        return FALSE
```

Testing the Event Checkers

1. Bumper Event Checker:
    a. Test Setup: Configure the test environment so you can manually trigger the front left bumper.
    b. Test Execution: Press and release the front left bumper to simulate the transition from BUMPER_NOT_TRIPPED to BUMPER_TRIPPED.
    c. Expected Outcome: The event checker should detect this change and post a bumper event. Verify this by checking the event queue or observing the system response (like an LED indicator or a log message).
2. Light Level Event Checker:
    a. Test Setup: Use a controllable light source to vary the light intensity around the sensor.
    b. Test Execution: Gradually increase the light intensity to cross the LIGHT_THRESHOLD and then decrease it to cross the DARK_THRESHOLD.
    c. Expected Outcome: The event checker should post a light level change event when crossing these thresholds. Validate the functionality by monitoring the events or changes in the system behavior (like an adaptive lighting response).
3. Battery Check Event:
    a. Test Setup: If possible, simulate different battery levels or use a variable power supply.

b. Test Execution: Vary the battery voltage to cross the BATTERY_LOW_THRESHOLD.

c. Expected Outcome: The event checker should detect the low battery condition and post a battery status change event. Check for notifications or system actions that respond to low battery events.

Modifications to ES_Configure.h

1. Add the prototypes of the new event checker functions to TemplateEventChecker.h
2. Add CheckBumperEvent, CheckLightLevel, and CheckBatteryEvent to the list to ensure they are called by the framework during its event checking phase. (ln76)
3. Add new event types (e.g., BUMPER_EVENT, LIGHTLEVEL_CHANGE, BATTERY_STATUS_CHANGE) to the enumeration of event types (ln 50)
4. Ensure that there are services configured to respond to the new events, ex:
    a. SERV_1_HEADER = "BumperService.h"
    b. SERV_1_INIT = InitBumperService
    c. SERV_1_RUN = RunBumperService
    d. SERV_1_QUEUE_SIZE = 3

Part 6 - Better Event Detection

```
# BETTER BUMPER EVENT CHECKER

Define DEBOUNCE_COUNT
Static variable bumperStateHistory[DEBOUNCE_COUNT]
Static variable lastBumperState

Function CheckBumperEvent() returns boolean:
     currentBumperState = Roach_ReadBumper()
     Update bumperStateHistory with currentBumperState
```

```
    if Consistent state in bumperStateHistory for
    DEBOUNCE_COUNT readings:
         if currentBumperState != lastBumperState:
               lastBumperState = currentBumperState
               PostEvent(BUMPER_CHANGE, currentBumperState)
               return TRUE
    return FALSE
```

```
# BETTER LIGHT LEVEL EVENT CHECKER

Define LIGHT_TO_DARK_THRESHOLD
Define DARK_TO_LIGHT_THRESHOLD
Static variable lastLightState

Function CheckLightEvent() returns boolean:
    currentLightLevel = Roach_LightLevel()
    if lastLightState is DARK and currentLightLevel >
DARK_TO_LIGHT_THRESHOLD:
        lastLightState = LIGHT
        PostEvent(LIGHT_LEVEL_CHANGE, LIGHT)
        return TRUE
    elif lastLightState is LIGHT and currentLightLevel <
LIGHT_TO_DARK_THRESHOLD:
        lastLightState = DARK
        PostEvent(LIGHT_LEVEL_CHANGE, DARK)
        return TRUE
    return FALSE
```

For the bumper event detection, test with manual presses and releases of the bumper, checking that the debounce logic correctly filters out noise.

For the light level detection, move the light source around the threshold levels to ensure that events are not spammed and hysteresis is functioning correctly.

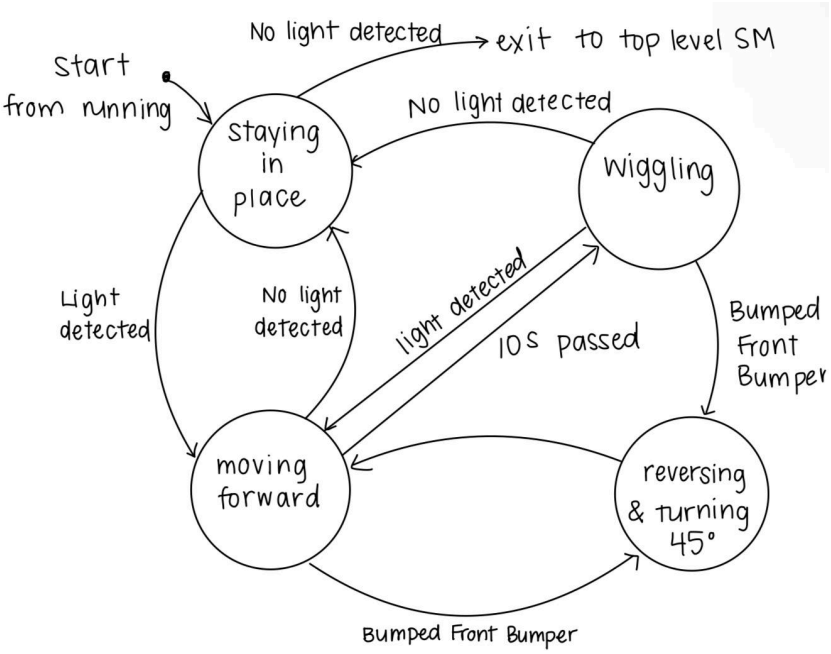
```
# Simple Bumper Service

function InitializeBumperService(Priority):
    Set service priority to Priority
    Post an ES_INIT event to the Bumper Service
    Start a periodic timer (BUMPER_SERVICE_TIMER) with a check
interval (BUMPER_CHECK_PERIOD)
    Return TRUE if successful, FALSE otherwise

function PostBumperService(Event):
    Post Event to Bumper Service queue using the stored priority
    Return result of post operation (TRUE or FALSE)

function RunBumperService(Event):
    Initialize ReturnEvent as ES_NO_EVENT

    Switch on Event type:

        case (ES_INIT):
               Perform any initialization needed for the bumper
               service

        case (ES_TIMEOUT):
               Call CheckBumperEvent function to check and post
               bumper events

               Restart the BUMPER_SERVICE_TIMER with
               BUMPER_CHECK_PERIOD

    Return ReturnEvent
```

Modifications to ES_Configure.h

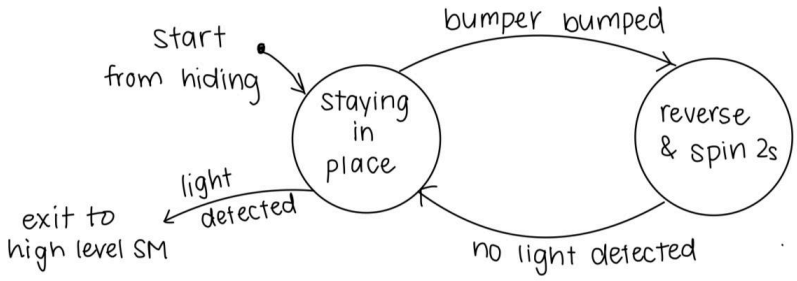
1. Service Registration:
    a. Adding the header file of the bumper service to the service list.
    b. Specify the InitializeBumperService and RunBumperService functions in the service configuration.
2. Timer Setup:

a. Assigning a timer (e.g., BUMPER_SERVICE_TIMER) in the framework's timer configuration section to manage the periodic execution of the bumper check. (200hz)

Part 7 - Finite State Machine



FSM for Running



FSM for Hiding

Helper Functions:

Reverse and Turn: The roach will reverse about one foot so all of its motors will be in backwards mode and then setting one wheel to turn.

Detecting light: The roach will continuously check the light levels and output TRUE when it had gone from light to dark

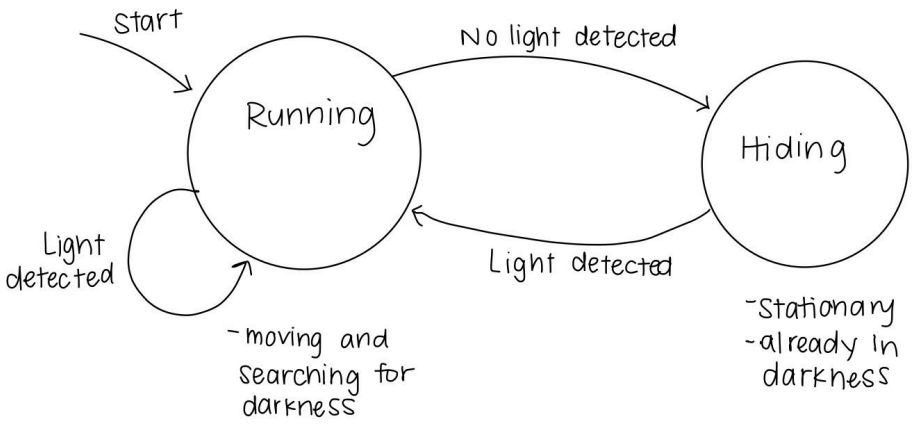
Detecting bumper hits: This function should output true if a bumper has been hit and then stop the motion of the roach

Moving forward: The roach will have the motors rotate at a steady speed

Wiggling: The roach will perform a wiggle motion moving side to side and end up facing a new direction. This means having one of the wheels steady and the other one moving and then doing the same to the other wheel. It will end up moving in a zig zag shape.

Reverse and Spin: the roach should move back for a short amount of time and then rotate (spin) so one wheel should be stationary while the other is moving.

Part 8 - Hierarchical State Machine



HSM for Roach

Each state has a state machines embedded in it which are referenced in part 7.