Lab 0 Prelab

Victor Perez Contreras

April 4, 2025

Contents

1 Part 1 - PCB Assembly and Soldering

1. Watch Videos
2. The iron should be set to either 360 or 420 degrees farenheit depending on if the material is lead or lead-free respectively.
3. The temperature has to be set accordingly as referenced above.
4. A hot weld will lead to discoleration of the material and bad spread of material at the joint. A cold weld will be weak and have very little discoleration.
5. Black in the weld means the heat was excessive leading to burning in the materials.

2 Part 2 - “Hello World!” on a roach

None.

3 Part 3 - Running the Roach Test Harness

None.

4 Part 4 - Roach Hardware Exploration

1. Read
2. Test Harness

- FLEFT_BUMP_MASK: Outputs and checks the current voltage level of the battery.
- FRIGHT_BUMP_MASK: Checks the roach's light sensors.
- RLEFT_BUMP_MASK: Cheks the left motor
- RRIGHT_BUMP_MASK: Cheks the right motor

3. Psuedocode for additional test harness:

```
void Motor_Test(void) {
    switch(keyboard_input) {
        case (w):
            // Move the roach forward
            break;
        case (a):
            // Move the roach to the left
            break;
        case (s):
            // Move the roach backwards
            break;
        case (d):
            // Move the roach to the right
            break;
        case (q):
            // Stop the test
            return;
    }
}
```

5 Part 5 - Event Detection

1. Pseudocode prototype of event checkers for the bump sensors and the light sensor.

```
// Note: Numbers are arbritary and will be adjusted accordingly for the implementation
#define BUMP_SENSOR_THRESH 100
#define LIGHT_SENSOR_THRESH 200
```

```c
int Check_Bump_Events(int prev_bump_state) {
    int curr_bump_state = Roach_ReadBumpers();

    // Detect an event in the bump
    if (curr_bump_state != prev_bump_state) {
        // Compare both the curr and prev bump state and return a new bump
        int new_bumps = compare(curr_bump_state, prev_bump_state);

        switch (new_bumps) {
            case FLEFT_BUMP_MASK:
                printf("Front Left Bumper hit!\n");
                // Back up and turn right
                break;
            case FRIGHT_BUMP_MASK:
                printf("Front Right Bumper hit!\n");
                // Back up and turn left
                break;
            case RLEFT_BUMP_MASK:
                printf("Rear Left Bumper hit!\n");
                // Pivot right
                break;
            case RRIGHT_BUMP_MASK:
                printf("Rear Right Bumper hit!\n");
                // Pivot left
                break;
            // Multiple sensors or no sensors
            default:
                if (new_bumps > 0) {
                    printf("Multiple bumpers hit: %d\n", new_bumps);
                    // Take some action
                }
                break;
        }
    }

    return curr_bump_state;
}
```

```c
int Check_Light_Events(int prev_light_state) {
    int light_reading = Roach_ReadLightSensor();
    int curr_light_state = (light_reading > LIGHT_SENSOR_THRESH) ? 1 : 0;

    // Detect an event in the light sensor
    if (curr_light_state != prev_light_state) {
        if (curr_light_state == 1) {
            printf("Light level rose above threshold: %d\n", light_reading);
            // Action for bright environment
```

```
        } else {
            printf("Light level fell below threshold: %d\n", light_reading);
            // Action for dark environment
        }
    }

    return curr_light_state;
}
```

2. Include a description of the modifications to ES Configure.h so that the test harness will run your event checkers.

In the ES_Configure.h file there is an enum ES_EventTyp_t and an array called EventNames. To include the new test harnesses the BUMP_SENSOR_TEST and LIGHT_SENSOR_TEST must be added to the objects. They should each be included as enum in the enum and a str for the array.

Additionally, prototypes for the functions must be included in the appropriate service section (1-5).

6 Part 6 - Better Event Detection

1. Pseudocode prototype for “better” event checkers for the bump sensors and the light sensor with debounce and hysteresis bounds.

```
// Bump Sensor Thresholds (Hz)
#define BUMP_SENSOR_THRESH 200
#define BUMP_SENSOR_LOW_THRESH 190
#define BUMP_SENSOR_HIGH_THRESH 210

// Time Variables
int time;       // (in ms)
#define CLK 100 // (in ms)

int Check_Bump_Events(int prev_bump_state) {
    // Verify we have waited a significant amount of time before
    if (TIMER < 100) {
        printf("Not enough time has passed since the last check.\n";
        return prev_bump_state;
    }

    int curr_bump_state = Roach_ReadBumpers();

    // Detect an event in the bump
    if (curr_bump_state != prev_bump_state) {
        // Compare both the curr and prev bump state and return a new bump
        int new_bumps = compare(curr_bump_state, prev_bump_state);
```

```
        switch (new_bumps) {
            case FLEFT_BUMP_MASK:
                printf("Front Left Bumper hit!\n");
                // Back up and turn right
                break;
            case FRIGHT_BUMP_MASK:
                printf("Front Right Bumper hit!\n");
                // Back up and turn left
                break;
            case RLEFT_BUMP_MASK:
                printf("Rear Left Bumper hit!\n");
                // Pivot right
                break;
            case RRIGHT_BUMP_MASK:
                printf("Rear Right Bumper hit!\n");
                // Pivot left
                break;
            // Multiple sensors or no sensors
            default:
                if (new_bumps > 0) {
                    printf("Multiple bumpers hit: %d\n", new_bumps);
                    // Take some action
                }
                break;
        }
    }

    // Resent the timer
    TIMER = 0;

    return curr_bump_state;
}
```

```
// Light Sensor Thresholds
#define LIGHT_SENSOR_THRESH 200
#define LIGHT_SENSOR_LOW_THRESH 200
#define LIGHT_SENSOR_HIGH_THRESH 200


int Check_Light_Events(int prev_light_state) {
    int light_reading = Roach_ReadLightSensor();
    int curr_light_state = (light_reading > LIGHT_SENSOR_THRESH) ? 1 : 0;

    // Detect an event in the light sensor
    if (curr_light_state != prev_light_state) {
        if (curr_light_state == 1) {
            printf("Light level rose above threshold: %d\n", light_reading);
```

```
            // Action for bright environment
        } else {
            printf("Light level fell below threshold: %d\n", light_reading);
            // Action for dark environment
        }
    }

    return curr_light_state;
}
```

2. Description of the modifications to ES Configure.h so that the test harness will run the new event checkers.

7 Part 7 - Finite State Machine (FSM)

1. Spend time to discuss your state machine with your partner. A well-named and labeled state machine diagram will save you hours of lab time. Test it out with your partner by imagining events and inputs and seeing what happens—before you code.
2. Create a good drawing of your FSM with all states and transitions labeled. A neat hand drawing or a program like Draw.io should work.
3. Create a list of the helper functions you think you will need, with a brief explanation of what they do (refer to Roach.h for examples).

8 Part 8 - Hierarchial State Machine (HSM)

1. Spend time to discuss your HSM4 with your partner. A well-named and labeled state machine diagram will save you hours of lab time. Test it out with your partner by imagining events and inputs and seeing what happens—before you code.
2. Create a good drawing of your HSM with all states (including sub-states) and transitions labeled. A neat hand drawing or a program like Draw.io should work