Lab 0 Roach

Rafael Delwart and Ryan Taylor
University of California, Santa Cruz
ECE118 Professor Gabriel Elkaim

4/5/2024

1 Overview

The primary objective of this lab is to familiarize us with the Event Service Framework through programming a robotic platform, the Roach, and developing essential soldering skills for electronic assembly. The lab is structured into eight parts, starting with the assembly and soldering of a minibeacon PCB, followed by a series of programming exercises that introduce us to embedded code development, event detection, finite state machines (FSM), and hierarchical state machines (HSM). Each section is designed to incrementally build on our understanding and skills in handling electronic components, writing and debugging embedded software, and effectively utilizing event-driven programming to control robotic behavior. Through the completion of these tasks, we are expected to gain practical experience in mechatronics, reinforcing theoretical knowledge with hands-on application. This report details the approach taken, challenges encountered, and solutions implemented across the lab exercises, providing a comprehensive overview of the learning outcomes achieved.

2 Part 1

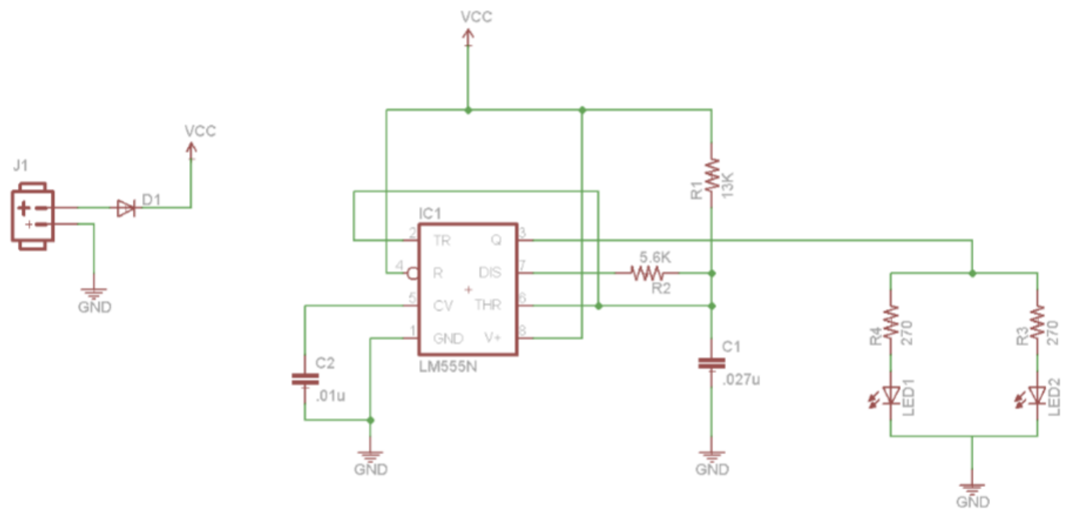

Figure 1: Circuit Diagram and Components List: The schematic features a power jack connector (J1), a diode (D1), an LM555N timer IC (IC1), resistors (R1 at 13KΩ, R2 at 5.6KΩ, R3 and R4 at 270Ω), capacitors (C1 at $0.027\mu F$, C2 at $0.01\mu F$, and light-emitting diodes (LED1 and LED2). The VCC and GND labels indicate the power supply connections, where the diode D1 provides polarity protection and the LM555N IC orchestrates the timing functions.

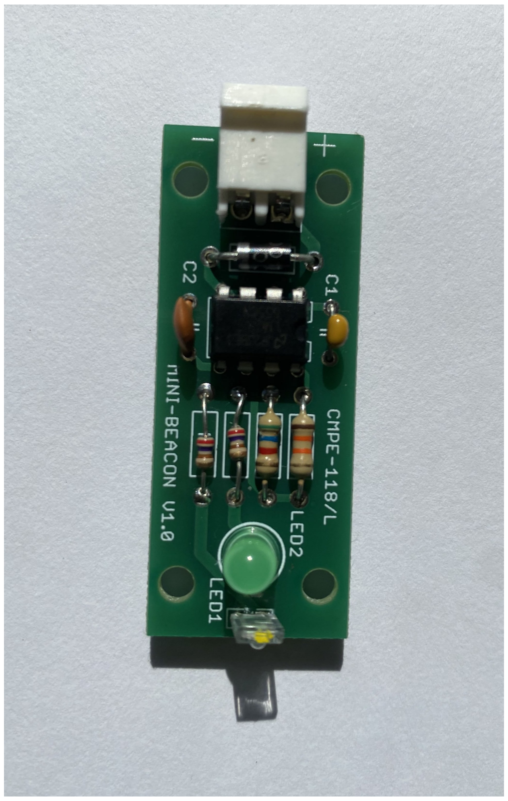

Figure 2: Image of the front of the fully assembled beacon

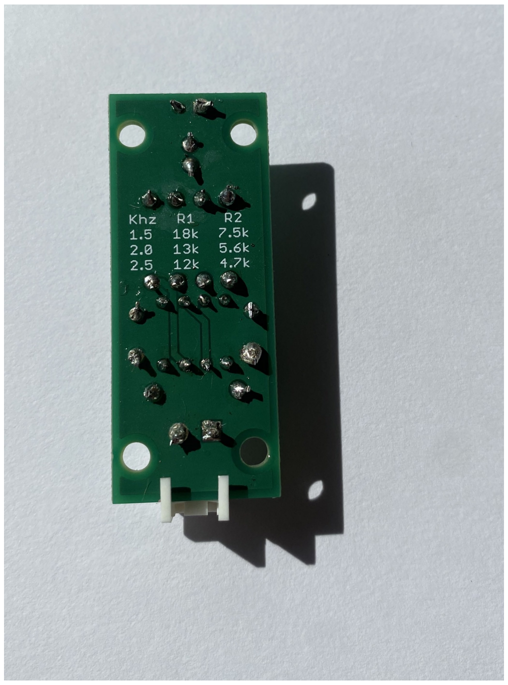

Figure 3: Image of the back of the fully assembled beacon

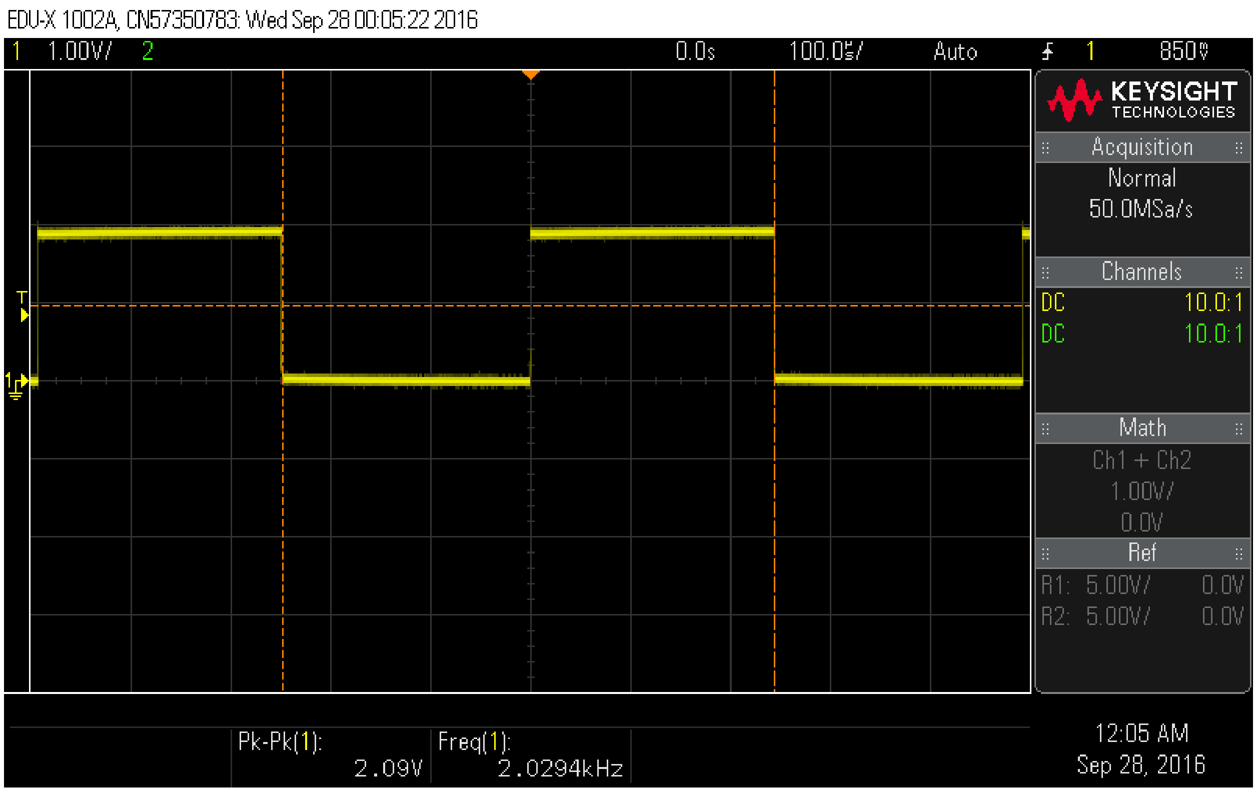

Figure 4: Oscilliscope capture of 2Khz square wave output of the beacon

ECE 118 Mini-Beacon board involved circuit design, construction, and allowed us to review many basic soldering skills. Seeing the LED light up and detecting the IR LED through a cellphone camera was a rewarding confirmation of our work. This not only honed our practical skills but also provided us with a vital piece of test equipment for detecting 2KHz IR signals in future labs. When measuring the frequency with an oscilloscope we examined a waveform with 2KHz frequency and amplitude of 2V as shown in figure 4. The one issue we ran into was connecting the scope probe to the circuit, we ended up probing the bottom of the circuit but next time we will be sure to level more clearance on the LED to allow for a probe to debug.

3 Part 2

Setting up embedded code on a new platform using MPLAB-X for our ECE118 project was a straightforward process but file pathing problems messed up a lot of our projects. When first launching MPLAB-X we created a new project, ensuring to select the PIC32MX320F128H processor, which matches our Uno32 boards. This precise selection was crucial for the compiler's behavior and the subsequent programming steps. We then navigated through configuring the PICKit3, selecting the XC32 compiler. We used GIT to facilitate collaborative work and for version control.

Then we added the necessary bootloader linker script, header files for BOARD, AD, and serial functionalities, along with their corresponding source files. This caused many issues, we weren't consistently choosing absolute files, additionally when sharing files between computers we needed to have all given files stored in our C drive. Finally, using the ds30Loader to flash our code onto the Uno32 and seeing the "Hello World!" output on the serial monitor demonstrated the successful setup of our embedded system. This process taught us the importance of precision in project configuration we ran into several pathing problems which could've been avoided with more careful file pathing.

4 Part 3

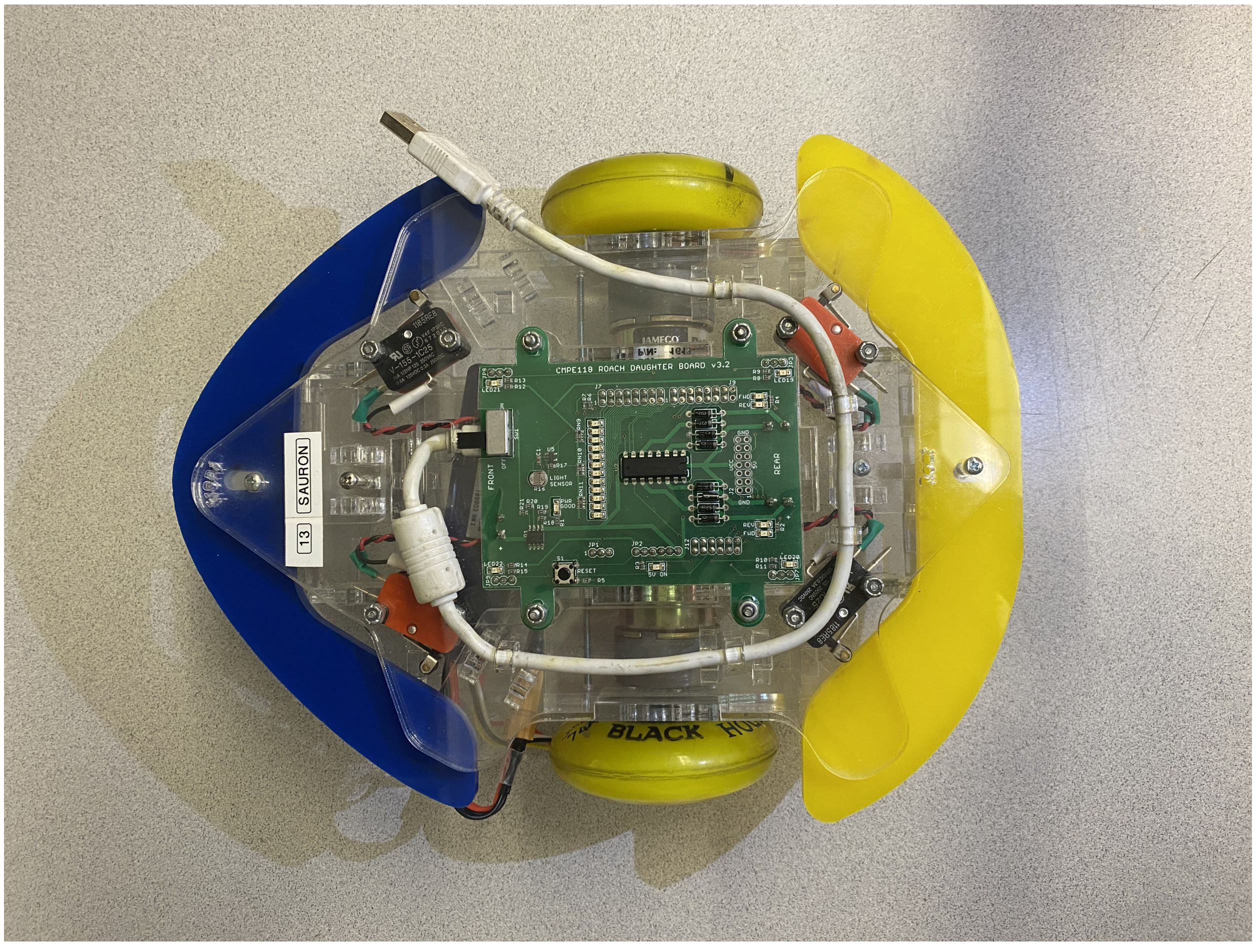

Figure 5: Image of the roach robot used for this lab, contains a PIC32 and several sensors and motors

During the evaluation of the robotic roach, we identified several potential issues. The most obvious issue was the disparity in the performance of the two motors. Our observations suggested that the variance in motor output could be attributed to the wear and tear experienced by the roaches over the years. Additionally, we noticed an inconsistency in motor strength between different roaches; some had more potent left motors, while others had stronger right motors. This inconsistency was also evident in the LDR sensors across different roaches; despite being exposed to the same intensity of light, the sensor readings varied.

Another observation was that the LDR sensors continued to produce readings even when the roach's motors were deactivated, in contrast to the bumpers, which did not generate any signal unless their switch was activated.

5 Part 4

To troubleshoot our robotic roach, we devised a rudimentary test harness consisting of a series of simple conditional statements within an infinite loop. When a frontal left bump was detected, we initialized the roach library. The testing of the left and right motors was triggered by a front right bump, utilizing the testRoachLeftMotor and testRoachRightMotor functions, respectively. The testing of the LEDs and the light sensor was initiated by a rear left bump, while the rear right bump was responsible for testing the bumper sensors. We encountered no significant challenges during this process, as our prior experience with implementing straightforward C scripts on the PIC32 for other courses made the task relatively straightforward.

This is the code for our simple test harness:

```
int main() {
    // Initialize Roach
    BOARD_Init();
    printf("Starting Tests");
    Roach_Init();
    // Check if any bumper is pressed to start the test
    while (1) {
        if (Roach_ReadFrontLeftBumper()) // if bumper 1 is pressed
        {
            printf("Bumper %d pressed.\r\n");
            printf("Starting tests...\r\n");
            test_Roach_Init();
            return 0;
        }
        if (Roach_ReadFrontRightBumper()) // if bumper 2 is pressed
        {
            printf("Bumper %d pressed.\r\n");
            printf("Starting tests...\r\n");
            test_Roach_LeftMtrSpeed();
            test_Roach_RightMtrSpeed();
            return 0;
        }
        if (Roach_ReadRearLeftBumper()) // if bumper 3 is pressed
        {
            printf("Bumper %d pressed.\r\n");
            printf("Starting tests...\r\n");
            test_Roach_LightLevel();
            test_Roach_BatteryVoltage();
            test_Roach_LEDSSet();
            test_Roach_LEDSGet();
            test_Roach_BarGraph();
            return 0;
        }
        if (Roach_ReadRearRightBumper()) // if bumper 4 is pressed
        {
            printf("Bumper %d pressed.\r\n");
            printf("Starting tests...\r\n");
            test_Roach_ReadFrontLeftBumper();
            test_Roach_ReadFrontRightBumper();
            test_Roach_ReadRearLeftBumper();
            test_Roach_ReadRearRightBumper();
            test_Roach_ReadBumpers();
            return 0;
        }
    }
    return 0; // Indicate success
}
// Implement test cases for each function

void test_Roach_Init(void) {
    // Test Roach_Init function
    Roach_Init();
    printf("Roach initialized.\r\n");
}

void test_Roach_LEDSGet(void) {
    uint16_t ledsState = Roach_LEDSGet();
    printf("Current LEDs state: 0x\%X\n", ledsState);
}

void test_Roach_LEDSSet(void) {
    Roach_LEDSSet(0xAAAA);
    printf("LEDS set to on \r\n");
}

void test_Roach_BarGraph(void) {
    Roach_BarGraph(6);
    printf("LED Bar Graph set to half \r\n");
}

void test_Roach_LeftMtrSpeed(void) {
    Roach_LeftMtrSpeed(-100); // Set left motor speed to 50
    printf("Left motor speed set to 50.\r\n");
}

void test_Roach_RightMtrSpeed(void) {
    Roach_RightMtrSpeed(-100); // Set right motor speed to -30
    printf("Right motor speed set to -30.\r\n");
}

void test_Roach_LightLevel(void) {
    unsigned int light_level = Roach_LightLevel();
    Roach_LEDSSet(light_level); // Set LEDs to light level
    printf("Current light level: %u\r\n", light_level);
}

void test_Roach_BatteryVoltage(void) {
    unsigned int battery_voltage = Roach_BatteryVoltage();
    Roach_LEDSSet(battery_voltage); // Set LEDs to battery voltage
    printf("Current battery voltage: %u\r\n", battery_voltage);
}

void test_Roach_ReadFrontLeftBumper(void) {
    unsigned char bumper_state = Roach_ReadFrontLeftBumper();
    printf("Front left bumper state: %u\r\n", bumper_state);
}

void test_Roach_ReadFrontRightBumper(void) {
    unsigned char bumper_state = Roach_ReadFrontRightBumper();
    printf("Front right bumper state: %u\r\n", bumper_state);
}

void test_Roach_ReadRearLeftBumper(void) {
    unsigned char bumper_state = Roach_ReadRearLeftBumper();
    printf("rear left bumper state: %u\r\n", bumper_state);
}

void test_Roach_ReadRearRightBumper(void) {
    unsigned char bumper_state = Roach_ReadRearRightBumper();
    printf("Rear right bumper state: %u\r\n", bumper_state);
}

void test_Roach_ReadBumpers(void) {
    unsigned char bumper_state = Roach_ReadBumpers();
    printf("Bumper state: %u\r\n", bumper_state);
}
```

6 Part 5

Our overarching approach was to systematically activate one event checker at a time. We began with the provided batteryChecker template as a foundation and replicated its structure to develop additional checkers for the bumpers and the LDR sensors. We opted to consolidate all bumper events into a single event, transmitting the specific bumper engaged via a 4-bit parameter. This method allowed for subsequent identification in our Finite State Machine (FSM) by interpreting the parameter values: 0x0001 for the front left bumper, 0x0010 for the front right, 0x0100 for the rear left, and 0x1000 for the rear right bumper.

During the debugging phase, we incorporated numerous print statements for monitoring and ensured that values were stored in appropriately typed variables. Despite meticulous type verification, we encountered multiple challenges. The most perplexing issue involved the LDR sensor consistently returning a value of 0xFFFF, regardless of the ambient light intensity. This anomaly was significant because the LDR sensor's maximum output should not exceed a 10-bit value, prompting a dive into the data type used for storing the sensor's output. We experimented with various data types, including unsigned int, unsigned long, and uint16, yet none resolved the issue. A breakthrough came when we realized that the roach library had not been initialized within our test harness; initializing it led to accurate sensor readings. The reason why the event checker functioned correctly for the bumpers but not for the LDR, despite both utilizing the roach library, remained elusive. Nonetheless, this discrepancy extended the duration of our debugging efforts significantly.

```
#define LIGHT_THRESHOLD 600
#define LIGHT_ON_THRESHOLD  (LIGHT_THRESHOLD)
#define LIGHT_OFF_THRESHOLD (LIGHT_THRESHOLD)

uint8_t LightChanged(void) {
    static ES_EventTyp_t lastEvent = INTO_DARK;
    ES_EventTyp_t curEvent;
    ES_Event thisEvent;

    uint8_t returnVal = FALSE;
    uint16_t lightVal = Roach_LightLevel();

    if (lightVal <= LIGHT_ON_THRESHOLD) // high light val means no light
    { // is light on?
        curEvent = INTO_LIGHT;
    } else if (lightVal > LIGHT_OFF_THRESHOLD) {
        curEvent = INTO_DARK;
    } else {
        curEvent = lastEvent;
    }
    if (curEvent != lastEvent) { // check for change from last time
        thisEvent.EventType = curEvent;
        thisEvent.EventParam = lightVal;
        returnVal = TRUE;
        lastEvent = curEvent; // update history
#ifndef EVENTCHECKER_TEST           // keep this as is for test harness
//        PostTemplateFSM(thisEvent);
                PostTemplateHSM(thisEvent);

#else
        SaveEvent(thisEvent);
#endif
    }
    return (returnVal);
}

uint8_t BumpedEvent(void) {
    static ES_EventTyp_t lastEvent = UNBUMPED;
    ES_EventTyp_t curEvent;
    ES_Event thisEvent;
    uint8_t returnVal = FALSE;
    uint16_t bumpers = Roach_ReadBumpers(); // read the battery voltage

    if (bumpers != 0) { // is battery connected?
        curEvent = BUMPED;
    } else {
        curEvent = UNBUMPED;
    }
    if (curEvent != lastEvent) { // check for change from last time
        thisEvent.EventType = curEvent;
        thisEvent.EventParam = bumpers;
        returnVal = TRUE;
        lastEvent = curEvent; // update history
#ifndef EVENTCHECKER_TEST           // keep this as is for test harness
//        PostTemplateFSM(thisEvent);
                PostTemplateHSM(thisEvent);
#else
        SaveEvent(thisEvent);
#endif
```

```
    }
    return (returnVal);
}
```

7 Part 6

In this section of our laboratory exercise, we focused on enhancing the responsiveness and reliability of our light and bump sensors through the implementation of hysteresis and debouncing techniques, respectively. The light sensors initially exhibited sensitivity issues, generating multiple events from a single transition from light to dark environments, contrary to the desired outcome of triggering a solitary event. To address this, we applied hysteresis by adjusting the threshold levels for light and dark, thereby facilitating smoother transitions without unnecessary event triggers. For the bump sensors, we introduced a debouncing mechanism using a straightforward service routine equipped with a basic timer. This routine comprises three primary functions: an initialization function, a posting function to relay events to the Finite State Machine (FSM), and a run function that embodies the debouncing logic. The core of our debouncing strategy involves invoking the bumper check logic exclusively in response to an ESTIMEOUT event. This approach ensures that each physical bumper interaction results in a single corresponding event being dispatched to the FSM, effectively eliminating the issue of multiple events stemming from a single bump.

```
#define LIGHT_THRESHOLD 600
// add hysteresis margins
#define LIGHT_ON_THRESHOLD  (LIGHT_THRESHOLD - HYSTERSIS)
#define LIGHT_OFF_THRESHOLD (LIGHT_THRESHOLD + HYSTERSIS)
#define HYSTERSIS 10

ES_Event RunTemplateService(ES_Event ThisEvent) {
    ES_Event ReturnEvent;
    ReturnEvent.EventType = ES_NO_EVENT; // assume no errors
    static ES_EventTyp_t lastEvent = UNBUMPED;
    ES_EventTyp_t curEvent;
    uint16_t bumpers = Roach_ReadBumpers(); // read the bumper states into a 4 bit value

    switch (ThisEvent.EventType) {
        case ES_INIT: //Nothing to setup
            break;
        case ES_TIMERACTIVE:
        case ES_TIMERSTOPPED:
            break;
        case ES_TIMEOUT:
            ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER, TIMER_0_TICKS);

            if (bumpers != 0) { // is bumper pressed
                curEvent = BUMPED;
            } else {
                curEvent = UNBUMPED;
            }
            if (curEvent != lastEvent) { // check for change from last time
                ReturnEvent.EventType = curEvent;
                ReturnEvent.EventParam = bumpers;
                lastEvent = curEvent;
#ifndef SIMPLESERVICE_TEST           // keep this as is for test harness
                PostTemplateHSM(ReturnEvent);
#else
                PostTemplateHSM(ReturnEvent);
#endif
            }
            break;
#ifdef SIMPLESERVICE_TEST     // keep this as is for test harness
        default:
            printf("\r\nEvent: %s\tParam: 0x%X",
                    EventNames[ThisEvent.EventType], ThisEvent.EventParam);
            break;
#endif
    }

    return ReturnEvent;
}
```

8 Part 7

Video of the FSM implemented on the roach: `https://drive.google.com/file/d/1rYtCG7T2h0A_m1H0rTW3j6Z5fSX1qgl6/view?usp=drive_link`

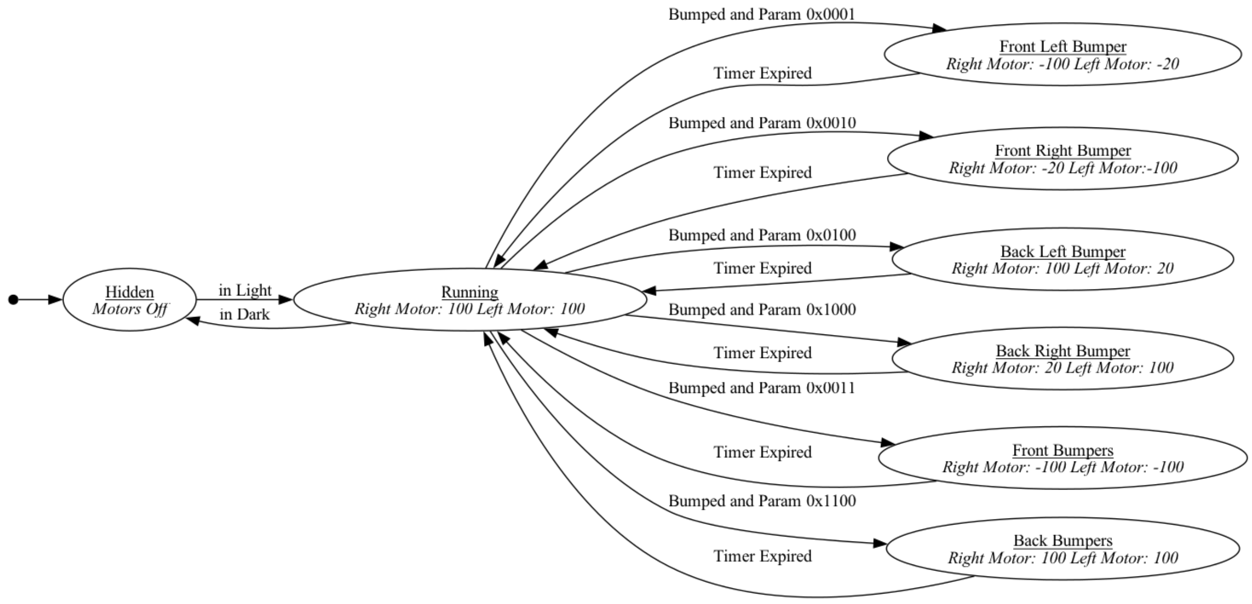

Figure 6: Roach basic Finite State Machine, transition between bumper state when bumpers pressed omitted for clearity

The code containing the logic for the FSM can be seen below, the conditions to move between bumper states have been omitted for brevity.

```
    case Hiding:
    //Turn off both motors to simulate hiding of a roach
    if (ThisEvent.EventType == ES_ENTRY) {
        Roach_RightMtrSpeed(0);
        Roach_LeftMtrSpeed(0);
    }
//If the roach enters the light it enters the running state
    if (ThisEvent.EventType == INTO_LIGHT) {
        nextState = Running;
        makeTransition = TRUE;
        ThisEvent.EventType = ES_NO_EVENT;
    }
    break;
case Running:
//Turn on both motors to simulate running of a roach
    if (ThisEvent.EventType == ES_ENTRY) {
        Roach_RightMtrSpeed(100);
        Roach_LeftMtrSpeed(100);
    }
//If the roach enters the dark it enters the hiding state
    if (ThisEvent.EventType == INTO_DARK) {
        nextState = Hiding;
        makeTransition = TRUE;
        ThisEvent.EventType = ES_NO_EVENT;
    }
//If bumped the roach should change direction based on where it hit
    if (ThisEvent.EventType == BUMPED) {
        //this code decodes which bumper was pressed and sends it the correct state
        //4 bit number 0x0001=FL 0x0010=FR 0x0100=BL 0x1000=BR
        if (ThisEvent.EventParam == 1) {
            nextState = FrontLeftBumper;
            makeTransition = TRUE;
            ThisEvent.EventType = ES_NO_EVENT;
        }
        if (ThisEvent.EventParam == 2) {
            nextState = FrontRightBumper;
            makeTransition = TRUE;
            ThisEvent.EventType = ES_NO_EVENT;
        }
        if (ThisEvent.EventParam == 4) {
            nextState = RearLeftBumper;
            makeTransition = TRUE;
            ThisEvent.EventType = ES_NO_EVENT;
        }
        if (ThisEvent.EventParam == 8) {
            nextState = RearRightBumper;
            makeTransition = TRUE;
            ThisEvent.EventType = ES_NO_EVENT;
        }
        //FL and FR  = 0x0011
        if (ThisEvent.EventParam == 3) {
            nextState = FrontBumpers;
            makeTransition = TRUE;
            ThisEvent.EventType = ES_NO_EVENT;
        }
        //BL and BR  = 0x1100
```

```
            if (ThisEvent.EventParam == 0xC) {
                nextState = RearBumpers;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }
        }
        break;
    case FrontLeftBumper:
        //set motors to back up and avoid left side
        if (ThisEvent.EventType == ES_ENTRY) {
            Roach_RightMtrSpeed(-100);
            Roach_LeftMtrSpeed(-20)
            //timer to back up
            ES_Timer_InitTimer(BUMP_TIMER, 2000);
        }
        //case to go back to dark while bumped
        if (ThisEvent.EventType == INTO_DARK) {
            nextState = Hiding;
            makeTransition = TRUE;
            ThisEvent.EventType = ES_NO_EVENT;
        }
        //once backed up continue to back to run state
        if (ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam == BUMP_TIMER) {
            nextState = Running;
            makeTransition = TRUE;
            ThisEvent.EventType = ES_NO_EVENT;
        }


        break;
    // same logic for the rest of the bumpers
    case FrontRightBumper:
        if (ThisEvent.EventType == ES_ENTRY) {
            Roach_RightMtrSpeed(-20);
            Roach_LeftMtrSpeed(-100);

            ES_Timer_InitTimer(BUMP_TIMER, 2000);
        }
        if (ThisEvent.EventType == INTO_DARK) {
            nextState = Hiding;
            makeTransition = TRUE;
            ThisEvent.EventType = ES_NO_EVENT;
        }
        if (ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam == BUMP_TIMER) {
            nextState = Running;
            makeTransition = TRUE;
            ThisEvent.EventType = ES_NO_EVENT;
        }



        break;
    case RearRightBumper:
        if (ThisEvent.EventType == ES_ENTRY) {
            Roach_RightMtrSpeed(40);
            Roach_LeftMtrSpeed(70);

            ES_Timer_InitTimer(BUMP_TIMER, 2000);
        }
        if (ThisEvent.EventType == INTO_DARK) {
            nextState = Hiding;
            makeTransition = TRUE;
            ThisEvent.EventType = ES_NO_EVENT;
        }
        if (ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam == BUMP_TIMER) {
            nextState = Running;
            makeTransition = TRUE;
            ThisEvent.EventType = ES_NO_EVENT;
        }

        break;
    case RearLeftBumper:
        if (ThisEvent.EventType == ES_ENTRY) {
            Roach_RightMtrSpeed(40);
            Roach_LeftMtrSpeed(70);

            ES_Timer_InitTimer(BUMP_TIMER, 2000);
        }
        if (ThisEvent.EventType == INTO_DARK) {
            nextState = Hiding;
            makeTransition = TRUE;
            ThisEvent.EventType = ES_NO_EVENT;
        }
        if (ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam == BUMP_TIMER) {
            nextState = Running;
            makeTransition = TRUE;
            ThisEvent.EventType = ES_NO_EVENT;
        }


        break;

    case RearBumpers:
        if (ThisEvent.EventType == ES_ENTRY) {
            Roach_RightMtrSpeed(100);
            Roach_LeftMtrSpeed(100);

            ES_Timer_InitTimer(BUMP_TIMER, 2000);
        }
        if (ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam == BUMP_TIMER) {
            nextState = Running;
            makeTransition = TRUE;
            ThisEvent.EventType = ES_NO_EVENT;
        }
        break;

    case FrontBumpers:
        if (ThisEvent.EventType == ES_ENTRY) {
            Roach_RightMtrSpeed(-100);
            Roach_LeftMtrSpeed(-20);

            ES_Timer_InitTimer(BUMP_TIMER, 2000);
        }
        if (ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam == BUMP_TIMER) {
            nextState = Running;
            makeTransition = TRUE;
            ThisEvent.EventType = ES_NO_EVENT;
        }
        break;
```

Our process for implementing the FSM was to code on state at a time at ensure robustness between states before continuing. We implemented all states first and then added timers to come back out of certain states after a certain delay. We ran into a few issues when creating the FSM, initializing new timers caused a little confusion, we were unsure which timer library we should employ, we first tried the simpler timer file which didn't work at all. We switched over to using timers similar to the one we used for the simple service. This proved to be far more effective. Another issue we ran into was properly initializing the ES Timers, we naively neglected to pass the timer to the FSM from the configure file. These issues slightly delayed our project, despite these setbacks this section of the lab took us the least amount of time. This portion of the lab was also the most satisfying, it finally allowed us to implement all of the previous code to actually manipulate the roach in real-time.

This FSM primarily operates within the Running state as long as there is light. Two main conditions trigger state changes. The first condition involves the BUMPED event, which functions more as a service checking the state of the ROACHREADBUMPERS every 5 ms. Upon receiving this event, we compare the Event Parameters with the bumpers to determine which bumpers were hit and transition to the corresponding state accordingly. All bumped states follow a similar pattern: initiating a 0.5-second timer and maneuvering in a pattern to evade obstacles before returning to the Running state.

The Running state and bumped states are influenced by the second main event, INTODARK. This event indicates a transition from light to darkness, allowing the FSM to enter the hiding state. In this state, the motors are deactivated, and bump detection is disabled. The only exit condition is receiving an INTOLIGHT event, which reactivates the motors, allowing the FSM to return to the Running state.

In summary, the FSM maintains its operation in the Running state under light conditions, responding to the BUMPED event to navigate obstacles and transitioning to the hiding state upon receiving the INTODARK event. The hiding state suspends motor activity and bump detection until the return of light triggers the transition back to the Running state.

9 Part 8

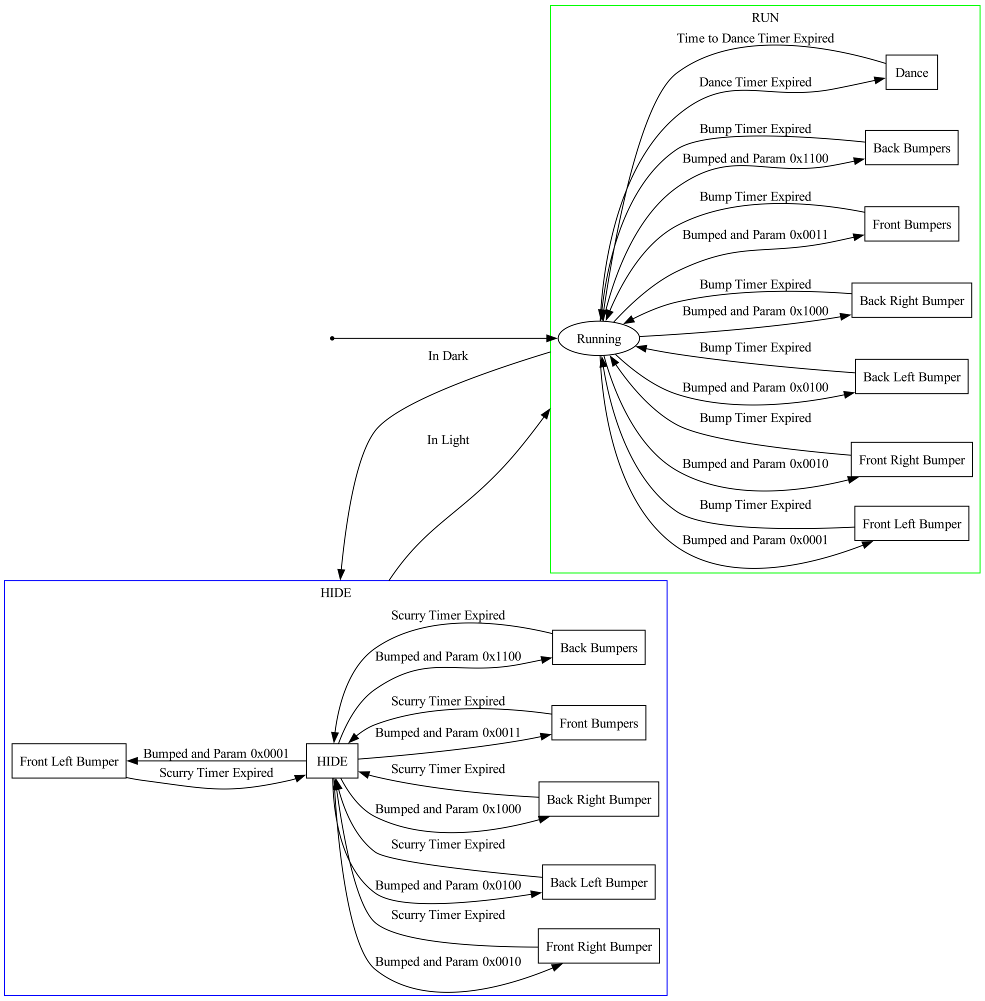

Figure 7: Roach Hierarchical State Machine, transition between bumper state when bumpers pressed omitted for clearity

Video of the HSM implemented on the roach: `https://drive.google.com/file/d/1X7fvnPPvSUgyDCO4CIbOfSrQX5066IRv/view?usp=drive_link`

The working strategy employed in this lab mirrored the approach explored in Part 7, emphasizing the modular design of the state machine and gradual implementation of features. We began by designing the top states for Hiding and Running, focusing on sending events to facilitate state transitions. This initial step enabled us to familiarize ourselves with the `ES_EVENTS` and `ESExit` functions integral to our design. Subsequently, we integrated sub-HSMs to test event propagation into nested states. The utilization of tattle-tale was pivotal during this phase, aiding in the construction of the HSM. This incremental feature implementation approach proved invaluable for debugging, allowing us to address issues more effectively as they arose.

The first bug we encountered stemmed from our event handling logic. Initially, we only directed events into the sub-state machine upon receiving a specific trigger, which resulted in unintended behavior. To rectify this, we revised our approach to consistently route events into the sub-state machine and then determine if they were relevant. If an event was consumed, we signaled this by sending an `ES No Event` back to the higher-level state machine, indicating that processing was complete and allowing progression to the next event in the queue.

One of the most challenging aspects of developing the HSM was managing timer restarts and determining where to handle `ES Timeouts`. This complexity arose particularly when multiple transitions relied on the same timer, leading to erroneous state transitions or unexpected jumps. Addressing this issue involved ensuring that timers unrelated to the current state were halted upon receiving an `ES Exit` event.

Reflecting on the project, I would prioritize implementing the state machines with a focus on `ES Entry` and `ES Exit` events. This approach, I believe, would enhance the clarity and intuitiveness of timer control while facilitating easier expansion into larger projects.

In summary, incorporating the HSM methodology enables deeper state machine development in a more manageable manner. The distinction in event handling compared to traditional FSMs affords greater flexibility and scalability for future endeavors.

Code for the HSM:

```
// State: Hiding
// This state manages the behavior of the system when it’s in the Hiding state.
case Hiding:
    // Run the sub-state machine for Hiding and update the event
    ThisEvent = RunHidingSubHSM(ThisEvent);

    // Handle events specific to the Hiding state
    switch (ThisEvent.EventType) {
        case ES_ENTRY:
            // On entry, set motor speeds for both right and left to 100
            Roach_RightMtrSpeed(100);
            Roach_LeftMtrSpeed(100);
            break;
        case INTO_LIGHT:
            // Transition to Running state on INTO_LIGHT event
            nextState = Running;
            makeTransition = TRUE;
            ThisEvent.EventType = ES_NO_EVENT; // Consume this event
            break;
        case ES_EXIT:
            // On exit, stop all timers (assuming timers 2 to 7 are related to this state)
            ES_Timer_StopTimer(2);
            ES_Timer_StopTimer(3);
            ES_Timer_StopTimer(4);
            ES_Timer_StopTimer(5);
            ES_Timer_StopTimer(6);
            ES_Timer_StopTimer(7);
            break;
        case ES_NO_EVENT:
        default:
            // No action for no event or default case
            break;
    }

// State: Running
// This state covers the Running behavior of the system.
case Running:
    // Run the sub-state machine for Running and update the event
    ThisEvent = RunRunningSubHSM(ThisEvent);
```

```
// Handle events specific to the Running state
switch (ThisEvent.EventType) {
    case ES_ENTRY:
        // No specific action on entry for Running state
        break;
    case INTO_DARK:
        // Transition to Hiding state on INTO_DARK event
        nextState = Hiding;
        makeTransition = TRUE;
        ThisEvent.EventType = ES_NO_EVENT; // Consume this event
        break;
    case ES_NO_EVENT:
        // No action for no event
        break;
    case ES_TIMEOUT:
        // Handle various timeout events to control motor speed and
        // transition through dance moves
        if (ThisEvent.EventParam == Dance_Timer) {
            // Initiate dance sequence with the first timer and set motor speeds
            ES_Timer_InitTimer(Dance_other, 1000);
            Roach_RightMtrSpeed(-100);
            Roach_LeftMtrSpeed(-100);
        }
        if (ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam == Dance_other) {
            Roach_RightMtrSpeed(40);
            Roach_LeftMtrSpeed(100);
            ES_Timer_InitTimer(Dance_second, 1000);
        }
        if (ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam == Dance_second) {
            Roach_RightMtrSpeed(-40);
            Roach_LeftMtrSpeed(-100);
            ES_Timer_InitTimer(Dance_third, 1000);
        }
        if (ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam == Dance_third) {
            Roach_RightMtrSpeed(100);
            Roach_LeftMtrSpeed(40);
            ES_Timer_InitTimer(Dance_fourth, 1000);
        }
        if (ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam == Dance_fourth) {
            Roach_RightMtrSpeed(-100);
            Roach_LeftMtrSpeed(-100);
            ES_Timer_InitTimer(Dance_fifth, 1000);
        }
        if (ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam == Dance_fifth) {
        // After the final dance step, transition back to Running state
            nextState = Running;
            makeTransition = TRUE;
            ThisEvent.EventType = ES_NO_EVENT; // Consume this event
        }
        break;
    case ES_EXIT:
        // On exit, stop all timers (assuming timers 2 to 7 are related to dance steps)
        ES_Timer_StopTimer(2);
        ES_Timer_StopTimer(3);
```

```
                    ES_Timer_StopTimer(4);
                    ES_Timer_StopTimer(5);
                    ES_Timer_StopTimer(6);
                    ES_Timer_StopTimer(7);
                    break;
                default:
                // No action for the default case
                break;
    }
            }
```

Code for the Running Sub-State Machine:

```
case Running: // in the first state, replace this with correct names

           switch (ThisEvent.EventType) {

               case ES_ENTRY: // only respond to ES_Init
                   Roach_RightMtrSpeed(100);
                   Roach_LeftMtrSpeed(100);
                   ThisEvent.EventType = ES_NO_EVENT;
                   ES_Timer_InitTimer(Dance_Timer, 5000);

                   break;
               case BUMPED:
                   if (ThisEvent.EventParam == 1) {
                       nextState = FrontLeftBumper;
                       makeTransition = TRUE;
                       ThisEvent.EventType = ES_NO_EVENT;
                       break;
                   }
                   if (ThisEvent.EventParam == 2) {
                       nextState = FrontRightBumper;
                       makeTransition = TRUE;
                       ThisEvent.EventType = ES_NO_EVENT;
                       break;
                   }
                   if (ThisEvent.EventParam == 4) {
                       nextState = RearLeftBumper;
                       makeTransition = TRUE;
                       ThisEvent.EventType = ES_NO_EVENT;
                       break;
                   }
                   if (ThisEvent.EventParam == 8) {
                       nextState = RearRightBumper;
                       makeTransition = TRUE;
                       ThisEvent.EventType = ES_NO_EVENT;
                       break;
                   }
                   if (ThisEvent.EventParam == 3) {
                       nextState = FrontBumpers;
                       makeTransition = TRUE;
                       ThisEvent.EventType = ES_NO_EVENT;
                   }
```

```
            if (ThisEvent.EventParam == 0xC) {
                nextState = RearBumpers;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }
            break;
        case ES_EXIT:
            ES_Timer_StopTimer(2);
            ES_Timer_StopTimer(3);
            ES_Timer_StopTimer(4);
            ES_Timer_StopTimer(5);
            ES_Timer_StopTimer(6);
            ES_Timer_StopTimer(7);
            break;

        default: // all unhandled events pass the event back up to the next level
            break;
    }
    break;

case FrontLeftBumper:

    switch (ThisEvent.EventType) {

        case ES_ENTRY:
            Roach_RightMtrSpeed(-100);
            Roach_LeftMtrSpeed(-50);
            ThisEvent.EventType = ES_NO_EVENT;
            ES_Timer_InitTimer(FrontLeftBumperTimer, 2000);
            ES_Timer_StopTimer(3);
            ES_Timer_StopTimer(4);
            ES_Timer_StopTimer(5);
            ES_Timer_StopTimer(6);
            ES_Timer_StopTimer(7);
            break;
            break;
        case ES_TIMEOUT:
            if (ThisEvent.EventParam == FrontLeftBumperTimer) {
                nextState = Running;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }

        case BUMPED:
            if (ThisEvent.EventParam == 1) {
                nextState = FrontLeftBumper;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
                break;
            }
            if (ThisEvent.EventParam == 2) {
                nextState = FrontRightBumper;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
                break;
            }
            if (ThisEvent.EventParam == 4) {
                nextState = RearLeftBumper;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
                break;
            }
            if (ThisEvent.EventParam == 8) {
                nextState = RearRightBumper;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
                break;
            }
            if (ThisEvent.EventParam == 3) {
                nextState = FrontBumpers;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }
            if (ThisEvent.EventParam == 0xC) {
                nextState = RearBumpers;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }
            break;
        case ES_EXIT:
            ES_Timer_StopTimer(FrontLeftBumperTimer);
            break;
        case ES_NO_EVENT:
            break;
        default:
            break;
    }
    break;

case FrontRightBumper:
    switch (ThisEvent.EventType) {
        case ES_ENTRY:
            Roach_RightMtrSpeed(-50);
            Roach_LeftMtrSpeed(-100);
            ThisEvent.EventType = ES_NO_EVENT;
            ES_Timer_InitTimer(FrontRightBumperTimer, 2000);
            ES_Timer_StopTimer(3);
            ES_Timer_StopTimer(4);
            ES_Timer_StopTimer(5);
            ES_Timer_StopTimer(6);
            ES_Timer_StopTimer(7);
            break;
        case ES_TIMEOUT:
            if (ThisEvent.EventParam == FrontRightBumperTimer) {
                nextState = Running;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }
        case BUMPED:
            if (ThisEvent.EventParam == 1) {
                nextState = FrontLeftBumper;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
                break;
            }
            if (ThisEvent.EventParam == 2) {
                nextState = FrontRightBumper;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
                break;
            }
            if (ThisEvent.EventParam == 4) {
                nextState = RearLeftBumper;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
                break;
            }
            if (ThisEvent.EventParam == 8) {
                nextState = RearRightBumper;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
                break;
            }
            if (ThisEvent.EventParam == 3) {
                nextState = FrontBumpers;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }
            if (ThisEvent.EventParam == 0xC) {
                nextState = RearBumpers;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }
            break;
        case ES_EXIT:
            ES_Timer_StopTimer(FrontRightBumperTimer);
            break;
        case ES_NO_EVENT:
            break;
        default:
            break;
    }
    break;

case RearLeftBumper:

    switch (ThisEvent.EventType) {

        case ES_ENTRY:
            Roach_RightMtrSpeed(40);
            Roach_LeftMtrSpeed(100);
            ThisEvent.EventType = ES_NO_EVENT;
            ES_Timer_InitTimer(RearLeftBumperTimer, 2000);
            ES_Timer_StopTimer(3);
            ES_Timer_StopTimer(4);
            ES_Timer_StopTimer(5);
            ES_Timer_StopTimer(6);
            ES_Timer_StopTimer(7);
            break;
        case ES_TIMEOUT:
            if (ThisEvent.EventParam == RearLeftBumperTimer) {
                nextState = Running;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }

        case BUMPED:
            if (ThisEvent.EventParam == 1) {
                nextState = FrontLeftBumper;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
                break;
            }
            if (ThisEvent.EventParam == 2) {
                nextState = FrontRightBumper;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
                break;
            }
            if (ThisEvent.EventParam == 4) {
                nextState = RearLeftBumper;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
                break;
            }
            if (ThisEvent.EventParam == 8) {
                nextState = RearRightBumper;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
                break;
            }
            if (ThisEvent.EventParam == 3) {
                nextState = FrontBumpers;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }
            if (ThisEvent.EventParam == 0xC) {
                nextState = RearBumpers;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }
            break;
        case ES_EXIT:
            ES_Timer_StopTimer(RearLeftBumperTimer);
            break;
        case ES_NO_EVENT:
            break;
        default:
            break;
    }
    break;

case RearRightBumper:

    switch (ThisEvent.EventType) {

        case ES_ENTRY:
            Roach_RightMtrSpeed(50);
            Roach_LeftMtrSpeed(100);
            ThisEvent.EventType = ES_NO_EVENT;
            ES_Timer_InitTimer(RearRightBumperTimer, 2000);
            ES_Timer_StopTimer(3);
            ES_Timer_StopTimer(4);
            ES_Timer_StopTimer(5);
            ES_Timer_StopTimer(6);
            ES_Timer_StopTimer(7);
            break;
        case ES_TIMEOUT:
            if (ThisEvent.EventParam == RearRightBumperTimer) {
                nextState = Running;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }

        case BUMPED:
            if (ThisEvent.EventParam == 1) {
                nextState = FrontLeftBumper;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
                break;
            }
            if (ThisEvent.EventParam == 2) {
                nextState = FrontRightBumper;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
                break;
            }
            if (ThisEvent.EventParam == 4) {
                nextState = RearLeftBumper;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
                break;
            }
            if (ThisEvent.EventParam == 8) {
                nextState = RearRightBumper;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
                break;
            }
```

```
                if (ThisEvent.EventParam == 3) {
                    nextState = FrontBumpers;
                    makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
                }
                if (ThisEvent.EventParam == 0xC) {
                    nextState = RearBumpers;
                    makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
                }
                break;
            case ES_EXIT:
                ES_Timer_StopTimer(RearRightBumperTimer);
                break;
            case ES_NO_EVENT:
                break;
            default:
                break;
        }
        break;

    case FrontBumpers:

        switch (ThisEvent.EventType) {

            case ES_ENTRY:
                Roach_RightMtrSpeed(-100);
                Roach_LeftMtrSpeed(-50);
                ThisEvent.EventType = ES_NO_EVENT;
                ES_Timer_InitTimer(FrontBumpersTimer, 2000);
                ES_Timer_StopTimer(3);
                ES_Timer_StopTimer(4);
                ES_Timer_StopTimer(5);
                ES_Timer_StopTimer(6);
                ES_Timer_StopTimer(7);
                break;
            case ES_TIMEOUT:
                if (ThisEvent.EventParam == FrontBumpersTimer) {
                    nextState = Running;
                    makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
                }

            case BUMPED:
                if (ThisEvent.EventParam == 1) {
                    nextState = FrontLeftBumper;
                    makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
                    break;
                }
                if (ThisEvent.EventParam == 2) {
                    nextState = FrontRightBumper;
                    makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
                    break;
                }
                if (ThisEvent.EventParam == 4) {
                    nextState = RearLeftBumper;
                    makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
                    break;
                }
                if (ThisEvent.EventParam == 8) {
                    nextState = RearRightBumper;
                    makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
                    break;
                }
                if (ThisEvent.EventParam == 3) {
                    nextState = FrontBumpers;
                    makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
                }
                if (ThisEvent.EventParam == 0xC) {
                    nextState = RearBumpers;
                    makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
                }
                break;
            case ES_EXIT:
                ES_Timer_StopTimer(FrontBumpersTimer);
                break;
            case ES_NO_EVENT:
                break;
            default:
                break;
        }
        break;
    case RearBumpers:

        switch (ThisEvent.EventType) {

            case ES_ENTRY:
                Roach_RightMtrSpeed(100);
                Roach_LeftMtrSpeed(100);
                ThisEvent.EventType = ES_NO_EVENT;
                ES_Timer_InitTimer(RearBumpersTimer, 2000);
                ES_Timer_StopTimer(3);
                ES_Timer_StopTimer(4);
                ES_Timer_StopTimer(5);
                ES_Timer_StopTimer(6);
                ES_Timer_StopTimer(7);
                break;
            case ES_TIMEOUT:
                if (ThisEvent.EventParam == RearBumpersTimer) {
                    nextState = Running;
                    makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
                }
            case BUMPED:
                if (ThisEvent.EventParam == 1) {
                    nextState = FrontLeftBumper;
                    makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
                    break;
                }
                if (ThisEvent.EventParam == 2) {
                    nextState = FrontRightBumper;
                    makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
                    break;
                }
                if (ThisEvent.EventParam == 4) {
                    nextState = RearLeftBumper;
                    makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
                    break;
                }
                if (ThisEvent.EventParam == 8) {
                    nextState = RearRightBumper;
                    makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
                    break;
                }
                if (ThisEvent.EventParam == 3) {
                    nextState = FrontBumpers;
                    makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
                }
                if (ThisEvent.EventParam == 0xC) {
                    nextState = RearBumpers;
                    makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
                }
                break;
            case ES_EXIT:
                ES_Timer_StopTimer(RearBumpersTimer);
                break;
            case ES_NO_EVENT:
                break;
            default:
                break;
        }
        break;
```