Lab 0 –The Roach Report

Talin Hallajian - Johnson Le

Part 1: PCB Assembly and soldering

In this part of the lab we assembled a mini-beacon which will be used in a later project. The mini-beacon is set to oscillate from 1.5-2.5 kHz. We each had to solder our own mini-beacons. Before soldering we made sure that the circuit was assembled correctly. We noticed that thinning a soldering iron before and after each solder attempt helps keep the iron tip from oxidizing which helps heat up the solder with less struggle. A good soldering job is can be represented by a volcano shape. There should not be any big solder balls or bridged connection between wires. The pictures below show our soldered mini-beacon. We did not have any bridged connections or big solder balls on our PCB board. To verify we build the mini-beacon correctly we powered it up and hooked it up to an o-scope. We observed different frequencies form each of our mini-beacons. There was a 2KHz frequency and 2.5kHz, which depended on the resistors we were given. An image of the 2kHz behavior on the o-scope is shown below. Our mini-beacons performed as expected.

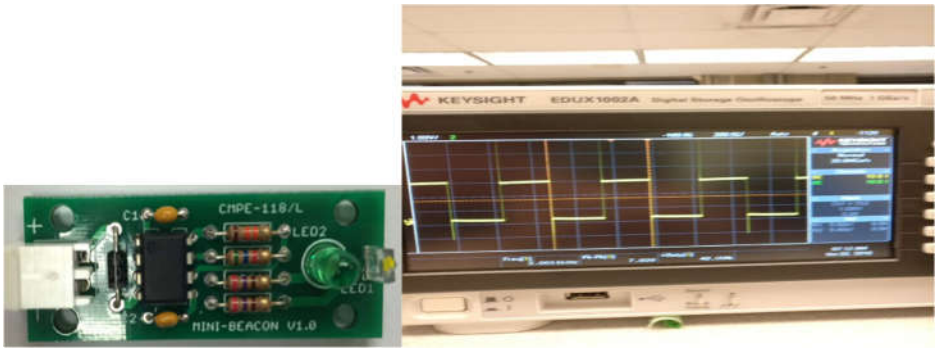

Figure 1 : Mini-beacon

Part 2: "Hello World!" On the ROACH

In this part of the lab we had to communicate with the microcontroller on the roach using MPLABX. Setting up the toolchain to make sure that we were able to communicate with the roach was very precise and tedious. We had to make sure that the right platforms were selected while creating the project in MPLABX. Once we were able to compile, we printed hello world on the serial port which verified that we were able to communicate with the microcontroller.

Part 3: Running the Roach test Harness

In this part of the lab, we verified that the roach hardware was working correctly by loading the test harness code given to us. Upon loading the test harness file, we noticed the LED bar flashed sequentially 5 times and then alternating LEDs 5 times. Pressing the front left

bumper displayed 7 volts for the battery. The front right displayed the current light level. Covering the sensor, we noticed the value was about 600 and without was about 300. The back bumpers were used to test the motors. We noticed that the left back bumpers started with flashing the LED bar 4 times. The amount of the LEDS on were dependent on the speed of the left motor starting with a high speed and decreasing till it stopped. The speed of the motor was displayed on the serial port. It then repeated going in reverse. The test ended with flashing the LED bar 4 times. The right back bumper had the same behavior but with the right motor and flashed 3 times at the start and end of the test. This was the right behavior were looking for which ensured that the roach was working properly and that we can move forward with the lab.

Part 4: Roach hardware exploration

In this part of the lab we created our own test harness to make sure we were able to send commands and read sensor values. We began with brainstorming with each other to discuss what tests will be run. This includes testing the bumpers, motors, LEDs, light sensor, and battery. The main difficulty was getting started and understanding how to use the provided functions from Roach. We tested each function individually to get an understanding before attempting to use them. Using a switch case setup, we used the 4 bumpers as triggers to begin the various tests as shown below. Information and instructions for running and using the tests are displayed on the serial port. The goal of this part of the lab was to detect any bugs with our roach by running our test.

```
while (1) {
    char bumperState = Roach_ReadBumpers();
    switch (bumperState) {
        case FrontLeftHit:
            FrontLeftTest();
            break;
        case FrontRightHit:
            FrontRightTest();
            break;
        case BackLeftHit:
            BackLeftTest();
            break;
        case BackRightHit:
            BackRightTest();
            break;
        default:
            break;
    }
}
```

Snippet 2: Switch statement of our tests.

Part 5: Event Detection

In this part of the lab we created event checkers for the bumpers being hit as well as change in light detected from the photo resistor. We condensed the four bumpers into one event because there is a limited amount of event queues. For the light sensor we had a light threshold of 600 and if we saw a change in the sensor reading from light to dark or dark to light we sent out an event to be serviced. The light sensor spammed out events because of noise around the threshold.

Starting out, we ran into some difficulties with building with the python script and importing some files. We kept getting errors with the python script due to having spaces in the directory. Moving the script to the desktop fixed the problem. When importing and copying the ES_Configure.h file into our project, we couldn't compile because it couldn't be found. It was somehow located inside of another project. The image below is a picture of our code to detect a change in the light sensor going from light to dark or dark to light.

```
uint8_t CheckLightDetection(void) {
    static ES_EventTyp_t lastLightEvent = INTO_DARK;
    ES_EventTyp_t currLightEvent;
    ES_Event lightEvent;
    unsigned int lightLevel = Roach_LightLevel();
    lightEvent.EventParam = lightLevel;
    uint8_t returnVal = FALSE;
    if (lightLevel > LIGHT_LEVEL_THRESHOLD) {
        currLightEvent = INTO_DARK;
    } else {
        currLightEvent = INTO_LIGHT;
    }
    if (currLightEvent != lastLightEvent) {
        lightEvent.EventType = currLightEvent;
        returnVal = TRUE;
        lastLightEvent = currLightEvent;
#ifndef EVENTCHECKER_TEST
        PostGenericService(lightEvent);
#else
        SaveEvent(lightEvent);
#endif
    }
    return (returnVal);
}
```

Snippet 3: Our event checker for the light sensor.

Part 6: Better Event Detection

In this part of the lab, we improved our event checkers by implementing debouncing with the bumpers and reducing noise from the light sensor. Since there is noise involved form reading the values from the photoresistor, we saw a spam of events when it switched from dark to light or light to dark. To fix this problem we used hysteresis bounds. This means we had two thresholds: a high and low bound. The gap between the two thresholds was held to be greater than the noise so that it was ignored when checking for events; since an event was only detected when the previous reading was below the lower threshold and the next reading was above the higher threshold (light to dark), or when the previous reading was read at above the higher threshold and the next reading was below the lower threshold (dark to light).

```
uint8_t CheckLightDetection(void) {
    static ES_EventTyp_t lastLightEvent = INTO_DARK;
    ES_EventTyp_t currLightEvent;
    ES_Event lightEvent;

    unsigned int currLightRead = Roach_LightLevel();
    uint8_t returnVal = FALSE;

    if (lastLightEvent == INTO_DARK && currLightRead <= LIGHT_THRESHOLD) {
        currLightEvent = INTO_LIGHT;
    } else if (lastLightEvent == INTO_LIGHT && currLightRead >= DARK_THRESHOLD) {
        currLightEvent = INTO_DARK;
    } else {
        currLightEvent = lastLightEvent;
    }

    if (currLightEvent != lastLightEvent) {
        lightEvent.EventType = currLightEvent;
        lightEvent.EventParam = currLightRead;
        returnVal = TRUE;
        lastLightEvent = currLightEvent;
#ifndef SIMPLESERVICE_TEST           // keep this as is for test harness
        PostGenericService(thisEvent);
#else
        SaveEvent(lightEvent);
#endif
#ifdef SIMPLESERVICE_TEST     // keep this as is for test harness
        printf("\r\nEvent: %s\tParam: %d",
                EventNames[lightEvent.EventType], lightEvent.EventParam);
#endif
    }
```

Snippet 4: Light detection with hysteresis

We also used debouncing to avoid spamming that can occur when the bumpers have been hit. This took a while because we did not understand how to create a service, but the ES_FRAMWORK helped clarify things. We used an ES_TIMOUT event every 200Hz to go and run the bumper service. This helped reduce bouncing that could have occurred while the sensor was running because it was not reading the sensor values continuously. We also had to fix the spamming of events at the transition point. To do this we stored 5 bumper readings in an array and only send out a bumper event if all the reading were the same, in tradeoff this made the detection of a bumper event slower. We tested our code by printing out the events that occurred on the serial port.

```
ES_Event RunBumperService(ES_Event ThisEvent) {
    ES_Event ReturnEvent;
    ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

    static unsigned char lastBumperRead = 0b0000;
    //    static ES_EventTyp_t lastEvent = BUMPER_RELEASE;
    ES_EventTyp_t currBumperEvent;
    unsigned char currBumperRead = Roach_ReadBumpers();

    switch (ThisEvent.EventType) {
        case ES_INIT:
            // No hardware initialization or single time setups, those
            // go in the init function above.
            //
            // This section is used to reset service for some reason
            break;
        case ES_TIMERACTIVE:
            break;
        case ES_TIMERSTOPPED:
            break;
        case ES_TIMEOUT:
            ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER, TIMER_0_TICKS);
            if (Debounce(currBumperRead)) {
                if (currBumperRead != lastBumperRead) {
                    if (currBumperRead > lastBumperRead) {
                        currBumperEvent = BUMPER_HIT;
                    } else if (currBumperRead < lastBumperRead) {
                        currBumperEvent = BUMPER_RELEASE;
                    }
                    lastBumperRead = currBumperRead;
                    ReturnEvent.EventType = currBumperEvent;
                    ReturnEvent.EventParam = currBumperRead;

#ifndef SIMPLESERVICE_TEST           // keep this as is for test harness
                    PostGenericService(ReturnEvent);
#else
                    PostBumperService(ReturnEvent);
#endif
                }
            }
            break;
#ifdef SIMPLESERVICE_TEST     // keep this as is for test harness
        default:
            printf("\r\nEvent: %s\tParam: %d",
                    EventNames[ThisEvent.EventType], ThisEvent.EventParam);
            break;
#endif
    }

    return ReturnEvent;
}
```

Snippet 5: Service for bumper event

```
uint8_t Debounce(unsigned char oldValue) {
    unsigned char myArray[5] = {0};
    int i;
    for(i = 0; i<4; i++)
    {
        myArray[i] = Roach_ReadBumpers();
    }
    int j;
    for(j = 0; j<3; j++)
    {
        if(myArray[j] != myArray[j+1])
        {
            return FALSE;
        }
    }
    return TRUE;
}
```

Snippet 6: Debouncer helper function

Part 7: Finite state machine

In this part of the lab we implemented the behavior of the roach using a finite state machine (FSM). The goal of state machine was to make the roach hide in the dark, run in the light and avoid obstacles in the way. We made sure to test as we go so that we can catch a bug easier and faster. We used the event checkers from the previous parts. We initially go to a hide state and then transition to a move forward state if the roach detects that it is in the light. Then depending on which bumper is hit the roach moves to get away from there. Instead of having blocking code we used timers to detect how long the roach should be pivoting away from the wall. The most difficult part was planning and testing the state machine to make sure that it behaved as intended. It was very important to plan the process on paper before implementing on code. We printed to the serial port as a debugging tool and used the LED's that were available on the microcontroller. The tattle tale helped us debug also as it showed which of the sates it went to.

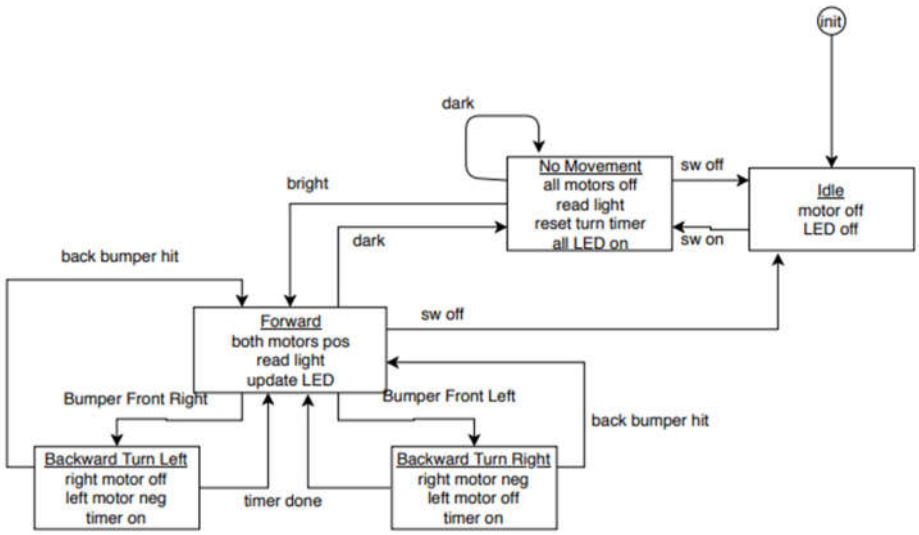

Figure 3 :FSM diagram

```
switch (CurrentState) {
    case FSMState_Init:
        /*Init: Not sure yet :D. Transition to idle*/
        if (ThisEvent.EventType == ES_INIT) {
            nextState = FSMState_Hide;
            makeTransition = TRUE;
            ThisEvent.EventType = ES_NO_EVENT;
            ES_Timer_SetTimer(TURN_TIMER, TURN_TICK);
            printf("\r\n");
        }
        break;
    case FSMState_Hide:
        /*Idle: Motors off. Read light. Reset Timer. ->Forward of bright.*/
        // lightState == Dark
        //read light level
        //if light, go to Forward state
        Roach_RightMtrSpeed(0);
        Roach_LeftMtrSpeed(0);
        switch (ThisEvent.EventType) {
            case INTO_LIGHT:
                nextState = FSMState_Forward;
                makeTransition = TRUE;
                Roach_BarGraph(0);
                break;
            case INTO_DARK:
                Roach_BarGraph(12);
                break;
```

```
        default:
            break;
    }

    break;
case FSMState_Forward:
    /*Forward: Read light. Moves till bump or dark*/
    //turn on motor on
    Roach_RightMtrSpeed(MOTOR_SPEED);
    Roach_LeftMtrSpeed(MOTOR_SPEED);
    switch (ThisEvent.EventType) {
        case INTO_LIGHT:
            break;
        case INTO_DARK:
            nextState = FSMState_Hide;
            makeTransition = TRUE;
            break;
        case BUMPER_HIT:
            Roach_RightMtrSpeed(MOTOR_OFF);
            Roach_LeftMtrSpeed(MOTOR_OFF);
            if (BUMPER_FRONT_RIGHT & ThisEvent.EventParam) {
                nextState = FSMState_Left;
                makeTransition = TRUE;
                ES_Timer_StartTimer(TURN_TIMER);
            } else if (BUMPER_FRONT_LEFT & ThisEvent.EventParam) {
                nextState = FSMState_Right;
                makeTransition = TRUE;
```

```
                ES_Timer_StartTimer(TURN_TIMER);
            }
        default:
            break;
    }
    break;
case FSMState_Left:
    //turn left. right off, left neg. if timer is done, trans to forward
    //if one/both back hits, trans to forward.
    Roach_RightMtrSpeed(MOTOR_OFF);
    Roach_LeftMtrSpeed(NEG_MOTOR_SPEED);

    if (ThisEvent.EventType == ES_TIMEOUT) {
        ES_Timer_StopTimer(TURN_TIMER);
        nextState = FSMState_Forward;
        makeTransition = TRUE;
        ES_Timer_SetTimer(TURN_TIMER, TURN_TICK);
    } else if (ThisEvent.EventType == BUMPER_HIT && ((ThisEvent.EventParam & BUMPER_BACK_LEFT) ||
(ThisEvent.EventParam & BUMPER_BACK_RIGHT))) {
        ES_Timer_StopTimer(TURN_TIMER);
        nextState = FSMState_Forward;
        makeTransition = TRUE;
        ES_Timer_SetTimer(TURN_TIMER, TURN_TICK);
    }
    break;
case FSMState_Right:
    Roach_RightMtrSpeed(NEG_MOTOR_SPEED);
    Roach_LeftMtrSpeed(MOTOR_OFF);
```

```
            if (ThisEvent.EventType == ES_TIMEOUT) {
                ES_Timer_StopTimer(TURN_TIMER);
                nextState = FSMState_Forward;
                makeTransition = TRUE;
                ES_Timer_SetTimer(TURN_TIMER, TURN_TICK);
            } else if (ThisEvent.EventType == BUMPER_HIT && ((ThisEvent.EventParam & BUMPER_BACK_LEFT) ||
(ThisEvent.EventParam & BUMPER_BACK_RIGHT))) {
                ES_Timer_StopTimer(TURN_TIMER);
                nextState = FSMState_Forward;
                makeTransition = TRUE;
                ES_Timer_SetTimer(TURN_TIMER, TURN_TICK);
            }
            break;
        default:
            break;
    }
    if (makeTransition == TRUE) {
        RunRoachFSM(EXIT_EVENT);
        CurrentState = nextState;
        RunRoachFSM(ENTRY_EVENT);
    }

    ES_Tail();
    return ThisEvent;
}
```

Snippet: 7: Our FSM implemented in code.

Part 8: Hierarchical state machine

In this part of the lab we implemented more difficult behaviors of the roach using a hierarchical state machine. Not only would the roach hide in the dark but also if the bumpers were hit it would go away from the bump for at least half a second. If the roach was in the light for more than five seconds, it will dance which helped the roach get unstuck in certain scenarios.

An HSM means that now states can have their own state machines (sub state machine). An event is sent all the way down to lowest state machine where the event is either taken care of(ES_NO_EVENT) and is sent back up or, nothing is done with the event and that event is send back up till it is absorbed. We brainstormed together for hours before getting into the code. In our top state machine, we had 2 states a hide that stops the motors and avoids a wall if a bumper is hit, and a move state which send the state to the sub state machine. In the sub state machine, we turn the roach depending on which bumper was hit, and we implement the dance state. We had a lot of problems communication between the two state machines and a lot of the mistakes came from forgetting to initialize the sub machine. We also had problems forgetting to start and stop the time at the appropriate times.

Our dance state spun the roach a little past 360 degrees so that it will end up in a different orientation than it was before which helped it get away from corners.

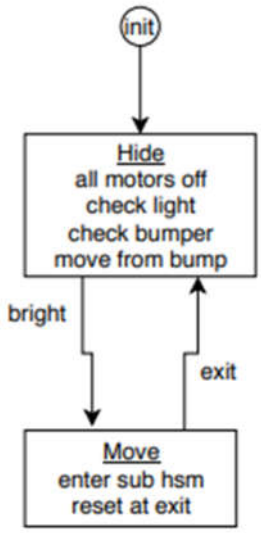

Figure 4: Top level HSM

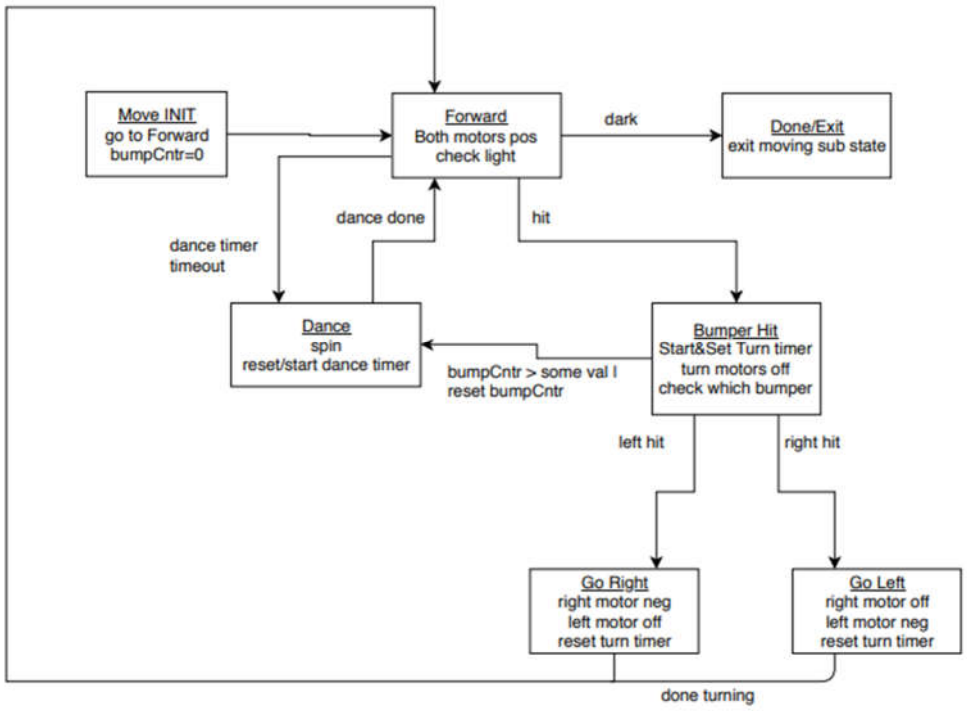

Figure 5: Lower level HSM

We had difficulties with implementing multiple timers and not letting them interfere with each other. For example, we had a timer that waits 5 seconds till the dance state is implemented and a timer that detected the duration of the dance which we kept interfering with each other. We solved most of our mistakes by making our code clean and having a lot defines. The tattle tale helped us debug by showing us which states it went to.

```
ES_Event RunRoachHSM(ES_Event ThisEvent) {
    uint8_t makeTransition = FALSE; // use to flag transition
    RoachHSMState_t nextState;

    ES_Tattle();

    switch (CurrentState) {
        case State_Init:
            if (ThisEvent.EventType == ES_INIT) {
                //init sub fsm when implemented
                InitAvoidSubHSM();
                nextState = State_Hide;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
                ES_Timer_SetTimer(DANCE_TIMER, DANCE_TICK);
                ES_Timer_StartTimer(DANCE_TIMER);

            }
            break;

        case State_Forward:
            if (ThisEvent.EventType == INTO_DARK) {
                nextState = State_Hide;
                makeTransition = TRUE;
                break;
            }
```

```
    ThisEvent = RunAvoidSubHSM(ThisEvent);

    switch (ThisEvent.EventType) {
        case ES_NO_EVENT:
            nextState = State_Hide;
            makeTransition = TRUE;
            ES_Event test;
            test.EventType = INTO_LIGHT;
            PostRoachHSM(test);
            break;
        default:
            break;
    }
    break;
case State_Hide:
    Roach_RightMtrSpeed(MOTOR_OFF);
    Roach_LeftMtrSpeed(MOTOR_OFF);
    switch (ThisEvent.EventType) {
        case INTO_LIGHT:
            nextState = State_Forward;
            makeTransition = TRUE;
            break;
        case BUMPER_HIT:
            ES_Timer_SetTimer(TURN_TIMER, HALFSEC_TICK);
            if (BUMPER_FRONT_RIGHT & ThisEvent.EventParam) {
                Roach_RightMtrSpeed(MOTOR_OFF);
                Roach_LeftMtrSpeed(NEG_SLOW_SPEED);
            }
            if (BUMPER_FRONT_LEFT & ThisEvent.EventParam) {
                Roach_RightMtrSpeed(NEG_SLOW_SPEED);
                Roach_LeftMtrSpeed(MOTOR_OFF);
            }
            if ((ThisEvent.EventParam & BUMPER_BACK_LEFT) || (ThisEvent.EventParam & BUMPER_BACK_RIGHT)) {
                Roach_RightMtrSpeed(SLOW_SPEED);
                Roach_LeftMtrSpeed(SLOW_SPEED);
            }
            break;
        case BUMPER_RELEASE:
            ES_Timer_StartTimer(TURN_TIMER);
            //start timer;
            break;
        case ES_TIMEOUT:
            ES_Timer_StopTimer(TURN_TIMER);
            break;
        default:
            break;
    }
    break;

default:
    break;
}
```

Our code for the top state machine. It has 2 states.

```
ES_Event RunAvoidSubHSM(ES_Event ThisEvent) {

    uint8_t makeTransition = FALSE; // use to flag transition
    AvoidSubHSMState_t nextState; // <- change type to correct enum

    ES_Tattle(); // trace call stack

    switch (CurrentState) {
        case AvoidSubState_Init:
            if (ThisEvent.EventType == ES_INIT) {
                nextState = AvoidSubState_Forward;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
                ES_Timer_SetTimer(TURN_TIMER, TURN_TICK);
            }
            break;
        case AvoidSubState_Forward:
            Roach_RightMtrSpeed(80);
            Roach_LeftMtrSpeed(80);

            switch (ThisEvent.EventType) {
                case INTO_DARK:
                    ThisEvent.EventType = ES_NO_EVENT;
                    break;
                case BUMPER_HIT:
                    ES_Timer_InitTimer(DANCE_TIMER, DANCE_TICK);    //this one to reset after bump
                    ES_Timer_InitTimer(DANCE_TIMER, DANCE_TICK);    //this one
                    Roach_RightMtrSpeed(MOTOR_OFF);
                    Roach_LeftMtrSpeed(MOTOR_OFF);
                    if (b >= 6) {
                        nextState = AvoidSubState_Dance;
                        makeTransition = TRUE;
                        ES_Timer_SetTimer(TURN_TIMER, 1700); //set how long to
                        ES_Timer_StartTimer(TURN_TIMER);
                        ES_Timer_StopTimer(DANCE_TIMER);
                        ES_Timer_SetTimer(DANCE_TIMER, DANCE_TICK);
                        b=0;
                    }
                    else if (BUMPER_FRONT_RIGHT & ThisEvent.EventParam) {
                        nextState = AvoidSubState_TurnLeft;
                        makeTransition = TRUE;
                        ES_Timer_StartTimer(TURN_TIMER);
                            b++;
                    } else if (BUMPER_FRONT_LEFT & ThisEvent.EventParam) {
                        nextState = AvoidSubState_TurnRight;
                        makeTransition = TRUE;
                        ES_Timer_StartTimer(TURN_TIMER);
                            b++;
                    }
                    printf("\r\n %d \r\n", b);
                    break;
              case ES_TIMEOUT:
                    //                    printf("\r\n dsajkkladjskadjls ljkas
                    Roach_RightMtrSpeed(MOTOR_OFF);
                    Roach_LeftMtrSpeed(MOTOR_OFF);
```

```
            ES_Timer_StopTimer(DANCE_TIMER);
            ES_Timer_SetTimer(DANCE_TIMER, DANCE_TICK);
            nextState = AvoidSubState_Dance;
            makeTransition = TRUE;
            ES_Timer_SetTimer(TURN_TIMER, 1000); //set how long to dance.
            ES_Timer_StartTimer(TURN_TIMER);
            break;
        default:
            break;
    }
    break;
case AvoidSubState_Dance:
    Roach_LeftMtrSpeed(-100);
    Roach_RightMtrSpeed(100);
    if (ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam == 1) {
        ES_Timer_StopTimer(TURN_TIMER);
        ES_Timer_SetTimer(TURN_TIMER, TURN_TICK);
        nextState = AvoidSubState_Forward;
        ThisEvent.EventType = ES_NO_EVENT;
        makeTransition = TRUE;
        ES_Timer_StartTimer(DANCE_TIMER);
    }
    break;
case AvoidSubState_TurnLeft:
    Roach_RightMtrSpeed(MOTOR_OFF);
```

```
            Roach_LeftMtrSpeed(NEG_MOTOR_SPEED);

            if (ThisEvent.EventType == ES_TIMEOUT || (ThisEvent.EventType == BUMPER_HIT && ((ThisEvent.EventParam &
BUMPER_BACK_LEFT) || (ThisEvent.EventParam & BUMPER_BACK_RIGHT)))) {
                ES_Timer_StopTimer(TURN_TIMER);
                nextState = AvoidSubState_Forward;
                ThisEvent.EventType = ES_NO_EVENT;
                makeTransition = TRUE;
                ES_Timer_SetTimer(TURN_TIMER, TURN_TICK);
            }
            break;
        case AvoidSubState_TurnRight:
            Roach_RightMtrSpeed(NEG_MOTOR_SPEED);
            Roach_LeftMtrSpeed(MOTOR_OFF);

            if (ThisEvent.EventType == ES_TIMEOUT || (ThisEvent.EventType == BUMPER_HIT && ((ThisEvent.EventParam &
BUMPER_BACK_LEFT) || (ThisEvent.EventParam & BUMPER_BACK_RIGHT)))) {
                ES_Timer_StopTimer(TURN_TIMER);
                nextState = AvoidSubState_Forward;
                ThisEvent.EventType = ES_NO_EVENT;
                makeTransition = TRUE;
                ES_Timer_SetTimer(TURN_TIMER, TURN_TICK);
            }
            break;
        default:
            break;
    }
```

Our sub state machine. That avoided walls and implemented the dance.

Youtube Video of our working Roach.

https://www.youtube.com/watch?v=N3yKNXylSTk

Our Checkoff Sheet