Lab 0 – The Roach Report

Justin Fortner & Alex Zuo

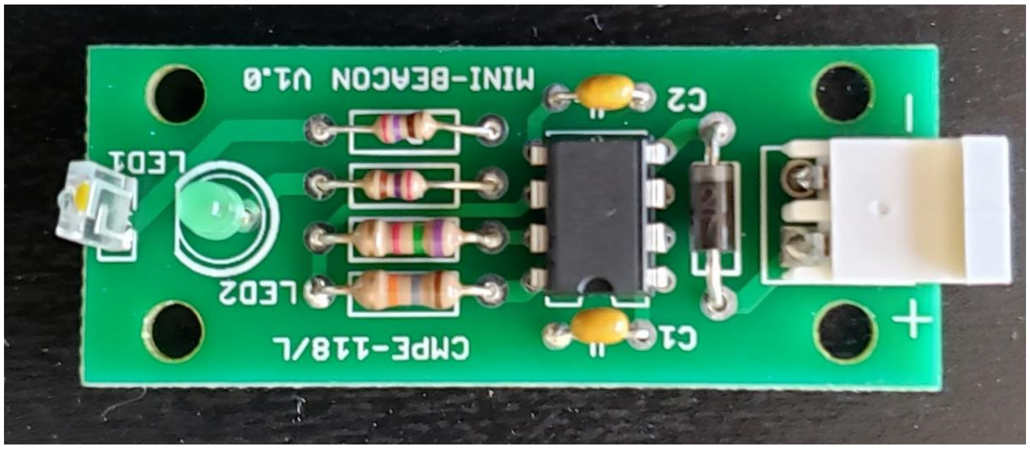
Summary:

This lab opens with a basic soldering task followed by a thorough introduction to the Roach. The Roach allows for the importance of concepts to be emphasized from the beginning. Some of these concepts include proper prelab completion, consult supplementary materials, time management, incremental coding and proper naming. The Roach also introduces ES_FRAMEWORK, FSMs, HSMs, event detection and services.

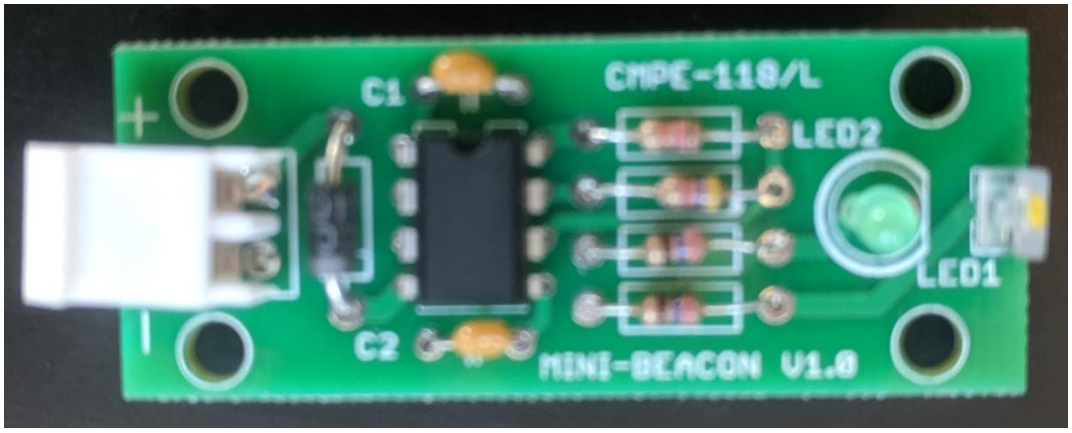
Part 1 – PCB Assembly and Soldering:

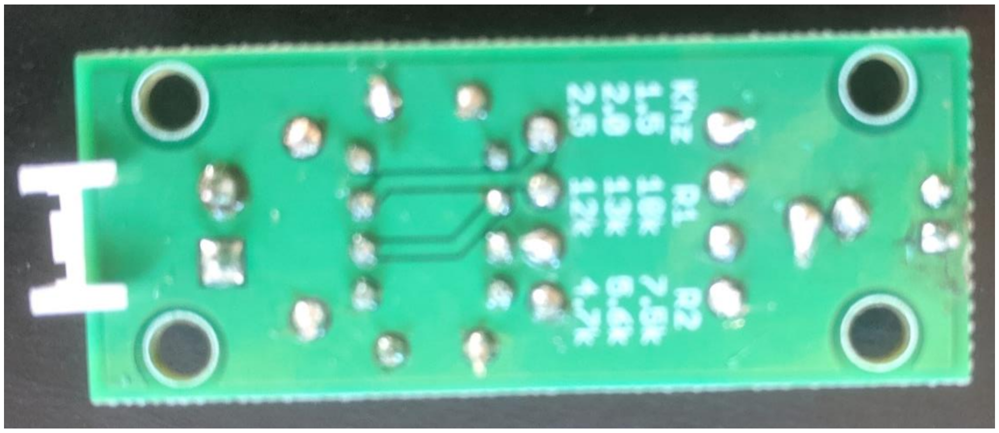
Part 1 requires the assembly and debugging of a minibeacon PCB. This task was easily and safely accomplished using the in-lab equipment. Goggles, fans and solder timers aid in a safe workspace. While the "helping hands", soldering iron and solder assist in solid solder connections. The soldering iron is locked at 680^{o}F so that the solder is heated properly and to minimize the chance of burning the flux in the core of the solder. After soldering completion, the entire project was checked for proper connections, short circuits, and loose solder balls.

Alex Zuo's Board:



Justin Fortner's Board:





Part 2 – "Hello World!" on a Roach:

Part 2 requires the "Hello World!" program to sun on the roach. This program will not take in any inputs from the roach but will display "Hello World!" in the serial window. The skills learned from this part are how to properly set up and program the roach.

These skills were learned, and the task was accomplished using the MPLABXNewProjectInstructions document. A summary of the process begins with creating a new project, selecting a Microchip Embedded Stand Alone Project, selecting the correct processor (PIC32MX320F128H), selecting the correct compiler (PICkit3), and name appropriately. Next, add in the appropriate header, source and loader files. Along with one properly named main file. The appropriate macros and file pathways are added through the drop-down menu at the top of the screen. This is necessary for MPLABX to find the included files. Macros can also be added in this window if necessary. Within the ds30Loader, the program used to upload the code to the roach, several settings need to be changed as well. These setting include the baud rate to be set to 115200, the reset to DTR with a time of 10, and the terminal to be set to switch to after write.

Part 3/4 – Running the Roach Test Harness/Roach Hardware Exploration:

Part 3/4 requires the creation of a test harness code for the Roach hardware. Our test harness uses the four bumpers to trigger four separate tests. The pairing are as follows: Front Left Bumper -> Wheels, Front Right Bumper -> Light Bar, Rear Left Bumper -> Light Sensor and Rear Right bumper -> Battery Voltage.

Wheel Test: The wheel test puts the wheels through a variety of different scenarios. Straight forward and straight backwards to test if the wheels are moving at the same rate. Tank turns to test if the wheels are isolated from one another. And slight turns to see how the roach behaves when in scenarios between a straight line and a tank turn.

Light Bar Test: The light bar test blinks each LED individually to show that they are individually addressable, followed by two full bar blinks. The two full blinks serve a dual purpose. The first showing that the light bar is group addressable and the second signaling the end of the test. This same double blink system signals the end of all tests.

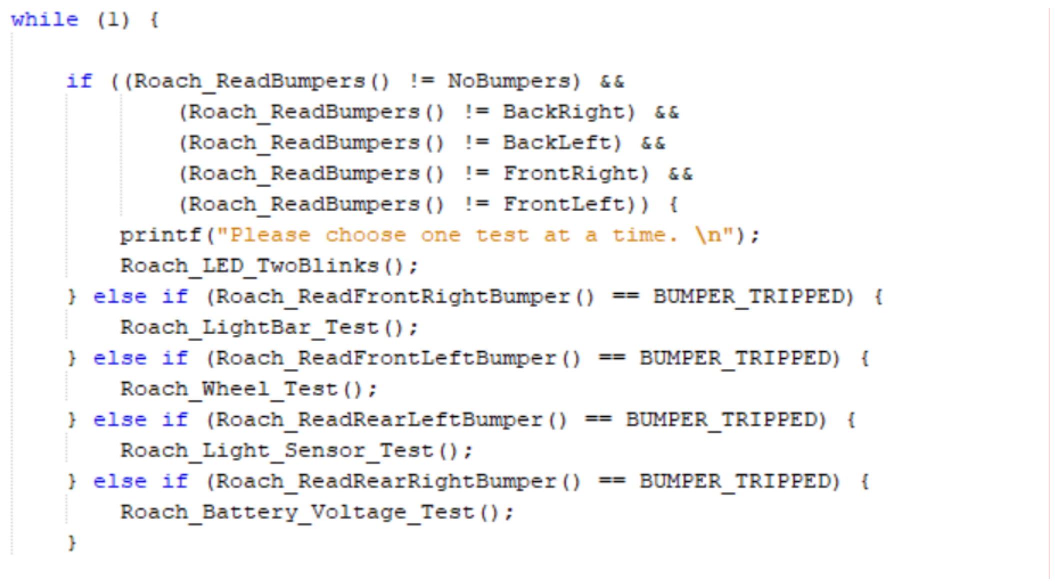
Light Sensor Test: The light sensor test run for roughly 10 seconds. Printing a light sensor reading to the serial window one a second. The end of the test is signaled by two full LED bar blinks.

Battery Voltage Test: The battery voltage test runs for roughly 10 seconds. Printing out the battery voltage every second. Unless the battery is below the critical 272 threshold. If this is the case the test is ended. Either ending is signaled with two full LED bar blinks.

Our largest initial difficulty was that we would spam our tests as there was no delay. Thus, we would get mangled text trying to print out light levels or voltage levels. This was solved by utilizing blocking for loops. This is not an ideal practice for more complex code, such as in part 5 on onward, but worked for our purposes here. Using helper functions we made a full second delay, half second, and a quarter second delay so that we had different breaks to fit the needs of our test harness. Our sudo code was very thorough so we had no further difficulties.

During our testing of several roaches we noticed two main things. Some of the roaches have motors that are reversed. Where a negative number applied to the motor will drive it forward while a positive will drive it backwards. The second observation made is that no two roaches run the same. This problem was overcome by making code extremely readable and adaptable to slight changes in hardware functionality. This robustness allowed us to adapt to whatever roach was at our workstation when we arrived in lab.

Code Snippets:



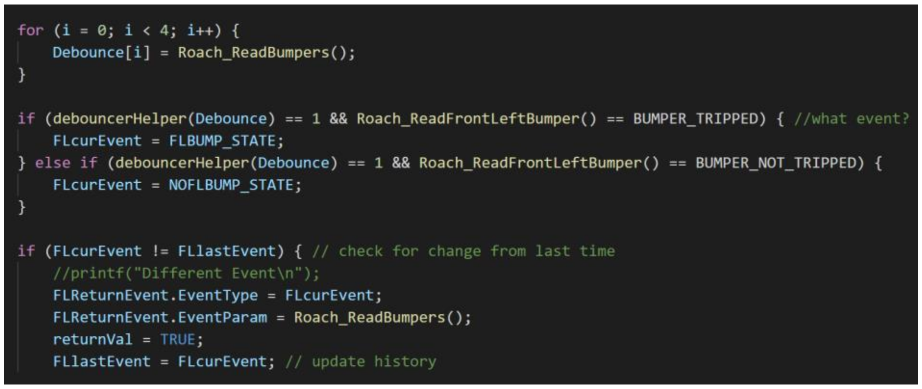
```
while (1) {

    if ((Roach_ReadBumpers() != NoBumpers) &&
            (Roach_ReadBumpers() != BackRight) &&
            (Roach_ReadBumpers() != BackLeft) &&
            (Roach_ReadBumpers() != FrontRight) &&
            (Roach_ReadBumpers() != FrontLeft)) {
        printf("Please choose one test at a time. \n");
        Roach_LED_TwoBlinks();
    } else if (Roach_ReadFrontRightBumper() == BUMPER_TRIPPED) {
        Roach_LightBar_Test();
    } else if (Roach_ReadFrontLeftBumper() == BUMPER_TRIPPED) {
        Roach_Wheel_Test();
    } else if (Roach_ReadRearLeftBumper() == BUMPER_TRIPPED) {
        Roach_Light_Sensor_Test();
    } else if (Roach_ReadRearRightBumper() == BUMPER_TRIPPED) {
        Roach_Battery_Voltage_Test();
    }
```

Part 5/6 – Event Detection/Better Event Detection:

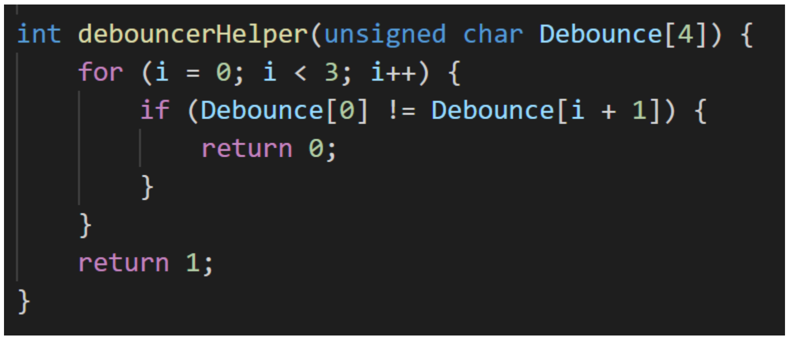
Part 5/6 requires the creation of event detection involving the bumpers and light detector. An event is defined as a meaningful change in measurement as provided by our hardware and determined by our software. The event checkers are simple non-blocking blocks of code that compare the current readings of our hardware to the previous reading and determine if this change is meaningful. If the change is deemed meaningful then a event is posted to the ES_FRAMEWORK and simple services.

Following the recommended reference material, we were able to successfully create an ES_FRAMEWORK project and run our event detectors. Our event detectors work as follows:

Bumper Event Checker: A timer is set to trigger the bumper event checker every 5ms. Four inputs are then read from all four bumpers. If all four inputs are equal, then we know the switch is no longer bouncing. As is one of the accepted debouncing strategies. The result is then compared to the previous bumper result and posts if there is a change. If there is no change or the switch is still bouncing, then no event is posted.

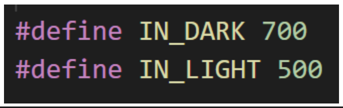


```
for (i = 0; i < 4; i++) {
    Debounce[i] = Roach_ReadBumpers();
}

if (debouncerHelper(Debounce) == 1 && Roach_ReadFrontLeftBumper() == BUMPER_TRIPPED) { //what event?
    FLcurEvent = FLBUMP_STATE;
} else if (debouncerHelper(Debounce) == 1 && Roach_ReadFrontLeftBumper() == BUMPER_NOT_TRIPPED) {
    FLcurEvent = NOFLBUMP_STATE;
}

if (FLcurEvent != FLlastEvent) { // check for change from last time
    //printf("Different Event\n");
    FLReturnEvent.EventType = FLcurEvent;
    FLReturnEvent.EventParam = Roach_ReadBumpers();
    returnVal = TRUE;
    FLlastEvent = FLcurEvent; // update history
```

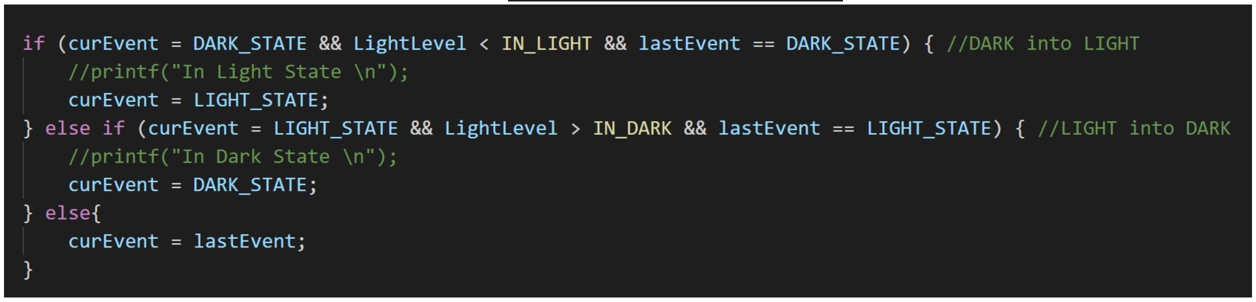


```
int debouncerHelper(unsigned char Debounce[4]) {
    for (i = 0; i < 3; i++) {
        if (Debounce[0] != Debounce[i + 1]) {
            return 0;
        }
    }
    return 1;
}
```

Light Event Checker: The light sensor is constantly being read and is able to post an event at any time. To avoid the spamming of events at every slight change in light we implemented hysteresis bounds. One bound to trigger a dark event when in a light event and one bound to trigger a light event when in dark. Through this strategy we were able to successfully detect significant changes and post only once change per event.



```
#define IN_DARK 700
#define IN_LIGHT 500
```



```
if (curEvent = DARK_STATE && LightLevel < IN_LIGHT && lastEvent == DARK_STATE) { //DARK into LIGHT
    //printf("In Light State \n");
    curEvent = LIGHT_STATE;
} else if (curEvent = LIGHT_STATE && LightLevel > IN_DARK && lastEvent == LIGHT_STATE) { //LIGHT into DARK
    //printf("In Dark State \n");
    curEvent = DARK_STATE;
} else{
    curEvent = lastEvent;
}
```

When first starting this portion, we had one initial difficulty. We failed to call Roach_Init() and were constantly getting a Roach_LightLevel() value of 65535. Once Roach_Init() was called our issue was solved and we were receiving proper light sensor values. As we progressed, we encountered two more difficulties. The first was getting our bumpers to work in the service project and the other was getting our lights to work in tandem to our service. Unbeknownst to us, we realized that our light sensor event checker was actually working the whole time, we just neglected to print out the event changes. The second issue we had is that we needed four different return event variables as we had a variable for each bumper. Initially, we had a single return variable, and this resulted in only one bumper being printed.

Part 7/8 – Finite State Machine/Hierarchical State Machine:

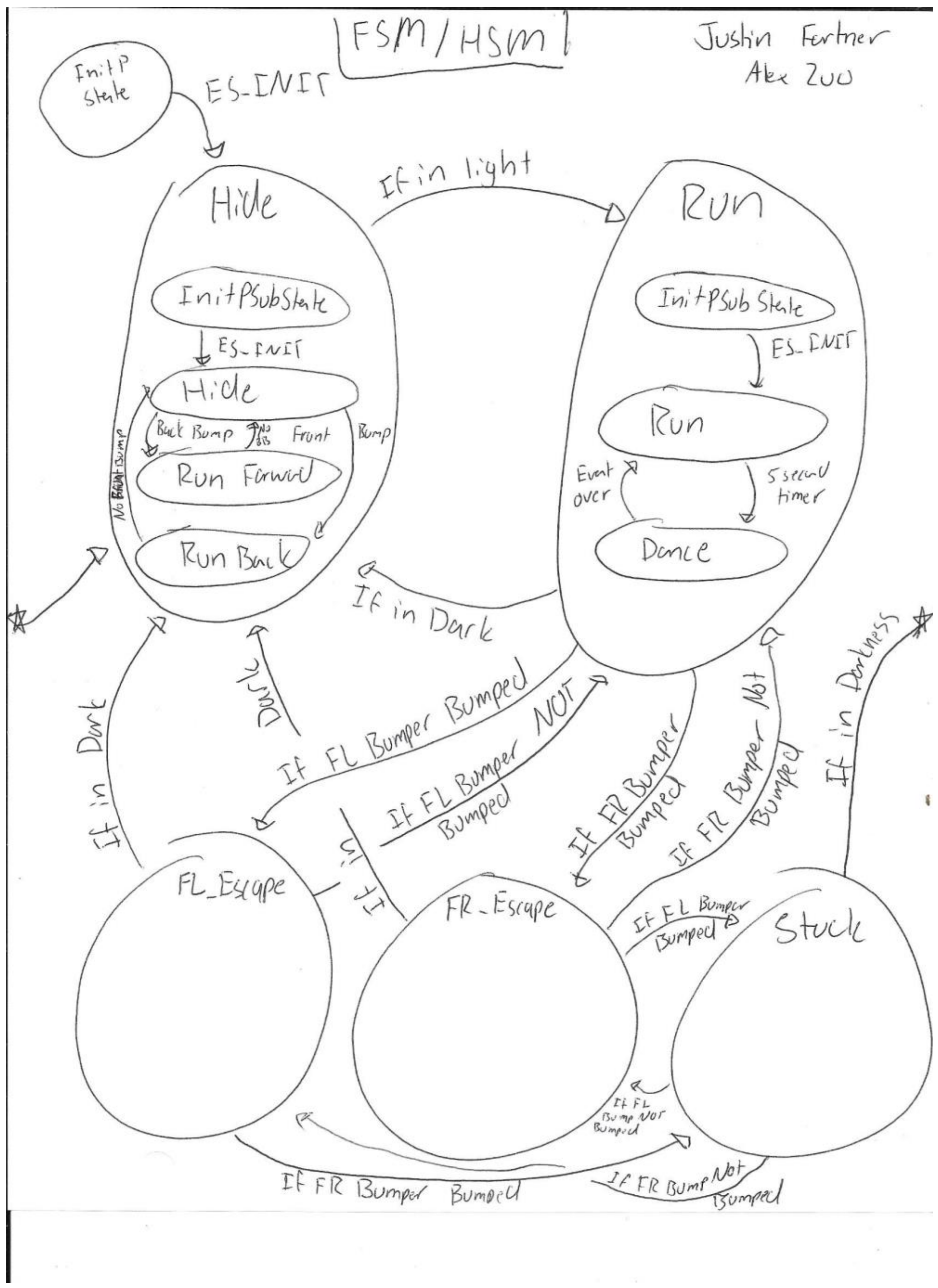
Part 5/6 requires the creation of Finite and Hierarchical state machines to implement the behavior of the Roach and reach the end of the obstacle course. Additional rules we must follow when implementing the state machine are: run from light, hide in darkness, don't get stuck, dance ever 5-10 seconds and if bumped and in darkness move away from the bump. Initial setup was accomplished by following the Creatin ES_Framework Projects document and using the associated templates. Code was also linked from previous parts of this project so that we did not have to duplicate our work. The time we spent during our prelab was rewarded greatly as our state machine worked flawlessly with very few tweaks. This was first verified by using the keyboard, followed by actual roach hardware inputs. And lastly tested in the obstacle course. An updated combined FSM and HSM diagram can be seen below. Two videos of it during the obstacle course and operating in dark conditions can be found https://youtu.be/nZUdaSORa-0 and https://youtu.be/Snl8tpWWNUo. The basic rundown is as follows.

The Roach starts in the hide state and checks for light. If it is in darkness it will remain still unless a bumper is pressed. If the front bumper is pressed the Roach will retreat. If the rear bumper is pressed the Roach will run forward.

If there is light, then it will begin to run in the Run state that is held in the Run HSM. If at any time in any state, the Roach finds darkness then it will return to the Hide state. If a bumper is pressed, then it will enter the FL_Escape or FR_Escape according to the bumper pressed. Within this state the Roach will turn away from the wall slightly. Once the bumper is not pressed it will return to run in the Run state that is held in the Run HSM. Effectively hugging a wall and following it until darkness is found.

If both bumpers get pressed at the same time the Roach will transition from FL_Escape or FR_Escape to Stuck state. In this state the roach will back up and reposition itself until it finds a wall where only one bumper is pressed. It will then return to bouncing between the FL_Escape or FR_Escape and the Run state.

Lastly, if at any time the Roach is not in darkness and the 5 second dance timer times out then the roach will do a dance.



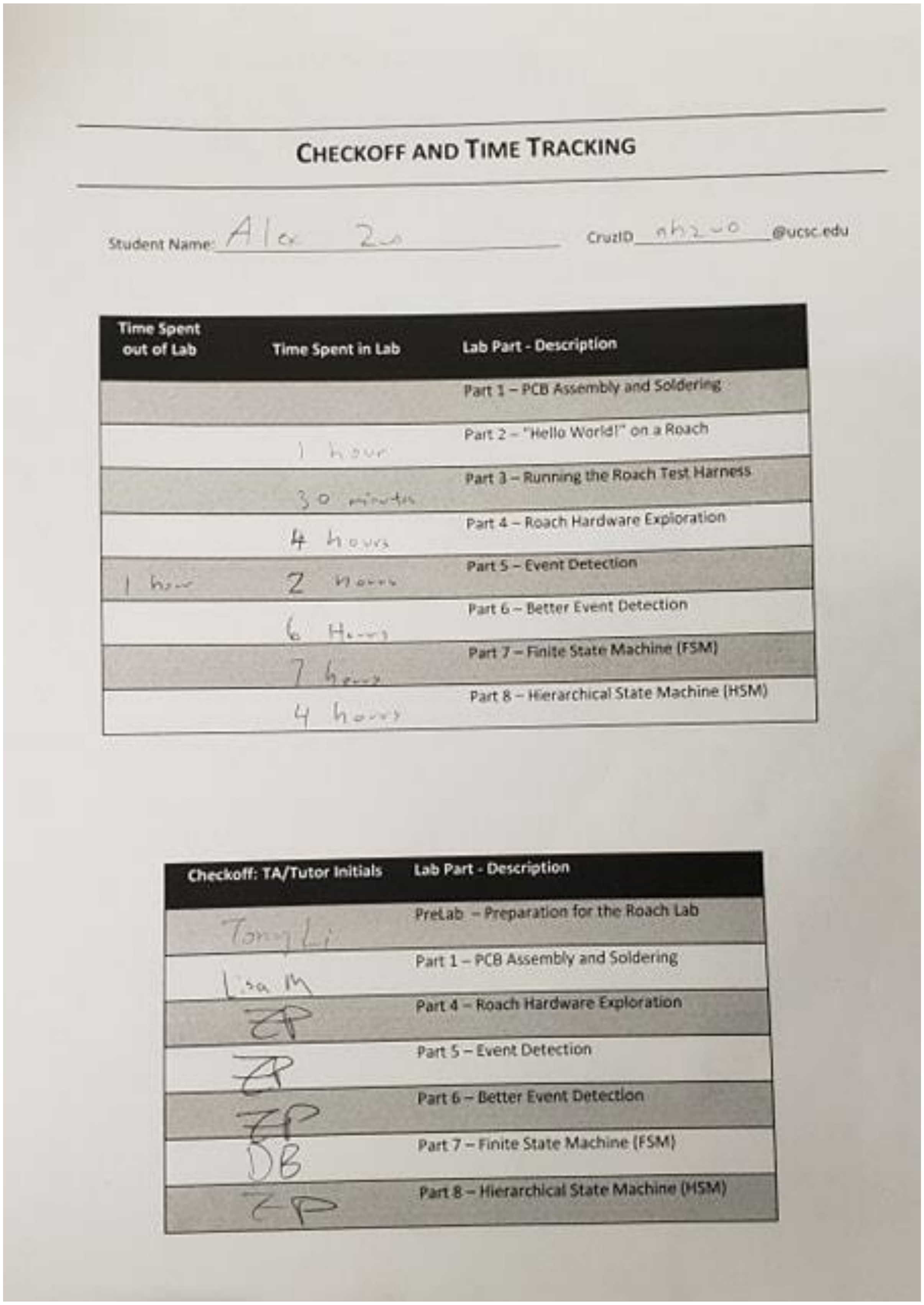
We had very few difficulties with this portion of the lab. The mist hindering difficulty we had come when setting up the FSM. We successfully got our state machine working with keyboard inputs. However, when attempting to get the same results with our roach hardware nothing was happening. After debugging we realized that we were still posting our events to the template, not the actual FSM. The keyboard would post to the FSM but not the roach. Once this kink was figured out it was just a matter of time and testing before our FSM and HSM were completed.

CHECKOFF AND TIME TRACKING

Student Name: Justin Fortner CruzID jfortner @ucsc.edu

Time Spent out of Lab	Time Spent in Lab	Lab Part - Description
0	1hr	Part 1 – PCB Assembly and Soldering
0	1hr	Part 2 – "Hello World!" on a Roach
0	1hr	Part 3 – Running the Roach Test Harness
0	5hrs	Part 4 – Roach Hardware Exploration
0	3hrs	Part 5 – Event Detection
0	7hrs	Part 6 – Better Event Detection
0	8hrs	Part 7 – Finite State Machine (FSM)
0	5hrs	Part 8 – Hierarchical State Machine (HSM)
	31hrs	

Checkoff: TA/Tutor Initials	Lab Part - Description
ZP	PreLab – Preparation for the Roach Lab
ZP	Part 1 – PCB Assembly and Soldering
ZP	Part 4 – Roach Hardware Exploration
ZP	Part 5 – Event Detection
ZP	Part 6 – Better Event Detection
OB	Part 7 – Finite State Machine (FSM)
ZP	Part 8 – Hierarchical State Machine (HSM)



CHECKOFF AND TIME TRACKING

Student Name: Alex Zu CruzID ahzuo @ucsc.edu

Time Spent out of Lab	Time Spent in Lab	Lab Part - Description
		Part 1 – PCB Assembly and Soldering
	1 hour	Part 2 – "Hello World!" on a Roach
	30 minutes	Part 3 – Running the Roach Test Harness
	4 hours	Part 4 – Roach Hardware Exploration
1 hour	2 hours	Part 5 – Event Detection
	6 Hours	Part 6 – Better Event Detection
	7 hours	Part 7 – Finite State Machine (FSM)
	4 hours	Part 8 – Hierarchical State Machine (HSM)

Checkoff: TA/Tutor Initials	Lab Part - Description
Tony Li	PreLab – Preparation for the Roach Lab
Lisa M	Part 1 – PCB Assembly and Soldering
ZP	Part 4 – Roach Hardware Exploration
ZP	Part 5 – Event Detection
ZP	Part 6 – Better Event Detection
DB	Part 7 – Finite State Machine (FSM)
ZP	Part 8 – Hierarchical State Machine (HSM)