Lab 0 – The Roach (FSMs and HSMs)

Overview:

In this lab you will program your first robot and solder up a small PCB you will use in later labs.

- Familiarize with the ES_FRAMEWORK and debugging tools.
- Program behavior of a small two-wheeled robot (the Roach).
- Concentrate on software using provided hardware and libraries.
- Learn basics of:
  - Event checkers
  - Test harnesses
  - Finite state machines
  - Hierarchical state machines
- Refresh basic soldering skills by assembling a small PCB.

Comments:

General Lab comments:

- Labs are time-consuming and challenging.
- Prepare by doing readings and prelab work.
- Start early and collaborate with your lab partner.
- Read the entire lab document thoroughly.
- Refer to supplementary readings for additional information.
- Seek clarification early to save debugging time.
- Mechatronics labs are open 24 hours/7 days

Lab checkoff:

- Check off each part with a tutor or TA to verify functionality and each team member's understanding.
- Strongly recommended to check off each part before moving to the next.

Lab0 specific comments:

- The mechatronics class has grown significantly, leading to limited equipment. Students must share resources and use sign-up sheets to reserve roaches.
- Be flexible with lab tasks; sections do not need to be completed in order
- All materials are provided, including workstations, roach robots, solder, soldering irons, and mini-beacon solder kits.
- Start the prelab early; it will save time during the lab.
- Submit the prelab early and be prepared to show it before checking out equipment.
- Teams may need to share roaches and other equipment. Please work together!

PreLab:

1. Choose a partner and join a group on CANVAS. Use Piazza to find partners if needed.
2. Get your mugshot taken by a TA during lab hours by Monday at 6pm.
3. Set up a shared folder (Google Drive or any other choice) or GitLab repository with your teammate.
4. Complete prelab exercises and submit a PDF file individually on CANVAS. Follow the requirements detailed in each section. Part1, Part4, Part5, Part6, Part7, Part8.

Part 1 – PCB Assembly and Soldering

Overview:

This assignment involves assembling and debugging one of the provided minibeacon PCBs. All necessary boards and components are included in your lab kit. For a quick soldering tutorial, refer to the **BELS Soldering Canvas Course**(you should have already received an invitation) or visit the SparkFun soldering tutorial.

Reference Material:

- ECE-118 Mini-Beacon documentation
- BELS Soldering Canvas Course
- Elecraft solder notes: Link

PreLab:

Answer the following questions:

1. Watch 3 or 4 videos on soldering technique (yes even if you know how to solder, it's a good refresher).
2. What temperature should you set the iron to?
3. What are the differences in how to handle Tin-Lead Vs Silver solder?
4. How can you tell a hot weld from a cold weld?
5. What does black in the weld mean?

Bumps and Road Hazards:

- Soldering irons are very hot; be careful not to burn yourself.
- Recently soldered items are still hot.
- Match the line on the PCB to the line on the diode body before soldering.
- De-soldering is a pain, though it can be done.

The puff of smoke you see when soldering is NOT lead, but the rosin core evaporating. The lead exposure from soldering is minimal and presents a very small hazard.

Tasks:

1. Become comfortable with soldering and read the ECE-118 Mini-Beacon documentation.
2. Each team member should solder one minibeacon PCB at different frequencies. **Odd-numbered** groups make 1.5kHz and 2kHz beacons; **even-numbered** groups make 2kHz and 2.5kHz beacons.
3. Inspect your board carefully for solder balls, bridged connections, or other issues.
4. Power it up and hook it up to an oscilloscope to verify the frequency.
5. Show your completed board to the TA or Tutor for check-off.
6. For the lab report, include a picture of the board you soldered and any observations about the experience.

Part 2 – "Hello World!" on a Roach

Overview:

This assignment is to get the basic "hello world!" code running on a roach. This will validate the toolchain and ensure that you can get output back from the roach on the serial port.

Reference Material:

- Tutorial: How to Start MPLAB X and ES_Framework
- Essential C – if you are unfamiliar or rusty with the C programming language

PreLab:

None.

Bumps and Road Hazards:

Put roaches up on blocks when on the tables to prevent wheel touching the surface. While the wheels should not spin in this part of the lab, it is essential you prevent them from driving off the lab benches in case you are wrong. Our roaches will NOT survive dropping off the benches onto the floor. They are hand-build, please treat them with respect.

Tasks:

1. Follow the steps in the MPLABX New Project Instructions document to get "Hello World!" working on the roach.
2. Validate that the entire toolchain is working, as is the roach.
3. Get checkoff from a TA/Tutor.
4. For the lab report, include a paragraph about getting embedded code up on the microcontroller of the roach.

Part 3 – Running the Roach Test Harness

Overview:

This assignment is to load the roach test harness code onto the roach, and verify that the hardware is all working correctly (note that this is an excellent thing to do every time you switch to a new roach).

Reference Material:

- Tutorial: How to Start MPLAB X and ES_Framework
- ECE-118 Roach Instructions
- CKO Chapters 1-5.

PreLab:

None.

Bumps and Road Hazards:

- **Put roaches up on blocks when on the tables to prevent wheel touching the surface.**
- Make sure your power switch is OFF before you load code.
- The wheels will turn during this test, ensure you do not drive the roach off the lab bench.

Tasks:

1. Follow the steps in the Test Harness section of the ECE118 Roach Instructions document.
2. Ensure that you see the correct behavior to validate the roach hardware and get checkoff from a TA/Tutor.
3. If your roach is not working, inform a TA/Tutor. **DO NOT** just put it back on the roach storage.
4. For the lab report, include a paragraph about testing the roach and what you observed.

Part 4 – Roach Hardware Exploration

Overview:

This assignment is to write your own test harness software using the Roach.h/.c module to test and explore the hardware, and get used to reading inputs and commanding outputs on the Roach. Essentially, you are writing your own test harness, or a set of helper functions that allow you to easily interface with the roach.

Reference Material:

- Tutorial: How to Start MPLAB X and ES_Framework
- ECE118 Roach Instructions
- Essential C – if you are unfamiliar or rusty with the C programming language
- CKO Chapters 1-5.

PreLab:

1. Read ECE118 Roach Instructions and the test harness section of the roach.c file below `#ifdef ROACH_TEST`.
2. Briefly describe what each test checks and outputs.
3. Create a pseudocode prototype of your own test harness for 4 additional tests you would like to have. You will use keyboard input as your test inputs.

Bumps and Road Hazards:

- **Put roaches up on blocks when on the tables to prevent wheel touching the surface.**
- Make sure your power switch is OFF before you load code.
- The wheels will turn during this test, ensure you do not drive the roach off the lab bench.

Tasks:

1. Create your own test harness code that uses keyboard input.
2. Test the following functionalities:
   - Move the roach forwards, backwards, and spins.
   - Read the bumpers (both individually and collectively).
   - Read the light sensor.
   - Control the LED bar.
3. Get checkoff from a TA/Tutor.
4. For the lab report, describe your strategy in the design and debugging of this test harness. Include code snippets of your test harness.

HINT: You may want to use the `GetChar()` function in serial.c/h. Develop your code incrementally, get one thing working first and then the next.

Part 5 – Event Detection

Overview:

This assignment is to write event detectors for bump events and light level detection. An event is a detectable **change** (e.g., INTO_LIGHT). Each event checker is a simple

subroutine, written to run very quickly. It will store a sensor reading and compare it to a previous reading. Significant differences will post an event to the framework. Simple event checkers may flood the framework with events. ("better" checker comes next part.)

Mastering event detectors is crucial for embedded projects.

Reference Material:

- Tutorial: How to Start MPLAB X and ES_Framework
- ECE118 Roach Instructions
- Essential C – if you are unfamiliar or rusty with the C programming language
- CKO Chapters 1-5.

PreLab:

1. Create a pseudocode prototype of your event checkers for the bump sensors and the light sensor.
2. Include a description of the modifications to ES_Configure.h so that the test harness will run your event checkers.

Bumps and Road Hazards:

Put Roaches up on blocks when on table. They should not be moving, but you don't want to realize that the hard way.

Tasks:

1. Build an ES_Framework project and event checkers using the template files in *ece118_template* folder in Canvas.
2. Write test harness code for your event checkers. You may not need a full ES_Framework (still needs ES_config.h) to run your event detectors and test harness.
   - You may use a project-defined pre-processor macro (e.g., `ROACH_EVENT_TEST`) to enable the test harness with conditional compilation. Activate the macro in the MPLABX project settings (the same as `#define` but in the compilation setting) to include the test harness code, and deactivate it to exclude the test harness code without modifying the source files.
3. Test light sensor and bumper event checkers. It is okay to have a lot of event spamming.
4. Get check off from a TA/Tutor.
5. For the lab report, describe your strategy in the design and debugging of your event checkers. Describe the difficulties of creating the code. Include code snippets of what you did.

Part 6 – Better Event Detection

Overview:

This assignment is to improve event detectors for bump sensors and light level detection. For bump sensors, implement a service to call the event detector at 200Hz and debounce the switches. For the light sensor, implement hysteresis bounds to prevent event spamming near the threshold.

You will need to set up a new service in the ES_Framework for the bumper part. This will be your introduction to using simple services, how to post and process events, and using the timers.

Reference Material:

- Tutorial: How to Start MPLAB X and ES_Framework
- ECE118 Roach Instructions
- Essential C – if you are unfamiliar or rusty with the C programming language
- CKO Chapters 1-5.

PreLab:

1. Create a pseudocode prototype of your "better" event checkers for the bump sensors and the light sensor with debounce and hysteresis bounds.
2. Include a description of the modifications to ES_Configure.h so that the test harness will run your event checkers.

Bumps and Road Hazards:

Put Roaches up on blocks when on table. They should not be moving, but you don't want to realize that the hard way.

Tasks:

1. Build an ES_Framework project using the template files in *ece118_template* folder in Canvas.
2. **Light Sensor:** Write event checkers and test harness code. You may not need a full ES_Framework (still ES_config.h only) to run your event detectors and test harness.

   In your event checker, implement hysteresis bounds for transitions between light and dark to improve robustness against small light level variations. Use the direction of change to select the appropriate threshold and track history with static variables to detect changes.
3. **Bump Sensor:** Write event checkers and test harness code. You will need a simple service ES_Framework project to debounce the bump sensors. Refer to the documentation (Creating ES Framework Projects) for a tutorial. The bump sensor

will use the ES_TIMEOUT events to run the service at 200Hz. Initialize and reset the timer each time.

Hint: Various debounce strategies can be implemented. One way is to create an array (or bits in an int) to store past readings. A valid method is to have a count "n" of the switch in the same state to be considered valid. There is always a tradeoff between responsiveness and robustness to bounces (and memory consumed). Arrays, masking, unions, and bitfields are useful here. Ensure your solution scales to handle all four switches simultaneously.

4. Test light sensor and bumper event checkers. Ensure that each event is detected only once when it occurs.
5. Get checkoff from a TA/Tutor. Show each test harness running.
6. For the lab report, describe your strategy in the design and debugging of your event checkers. Describe the difficulties of creating the code. Include code snippets of what you did (only your edited functions).

Part 7 – Finite State Machine

Overview:

This assignment is to write the finite state machine (FSM) that implements the behavior of a cockroach on your robot. The canonical rules are: **"run from light, hide in darkness, don't get stuck."**

You will use all the code you have already generated and implement it using the ES_Framework.

Reference Material:

- HSM Supplementary Lecture Video
- ElectricToothbrushFSM
- Tutorial: How to Start MPLAB X and ES_Framework
- ECE118 Roach Instructions
- Essential C – if you are unfamiliar or rusty with the C programming language
- CKO Chapters 1-5.

PreLab:

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.[1] Test it out with your partner by imagining events and inputs and seeing what happens—before you code.[2]
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.

[1] Think a **3-5x** improvement in your lab efficiency.

[2] This process, known as "walking through your code," is invaluable for identifying corner cases and missing state/event pairs early, making debugging easier. Addressing potential bugs before coding can save significant time and effort later.

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).

Bumps and Road Hazards:

Put Roaches up on blocks when on table. Do **NOT** ever run your roach live on the lab benches. If they are moving, they go on the floor. We don't trust you to catch them when they roll off the benches.

Tasks:

1. Set up an ES_Framework project and use the FSM templates to create a simple state machine. Utilize your event detectors from Parts 6. Build your project incrementally, building from previous parts without copying them to a new project.[3]
2. Implement the roach behavior based on your state machine diagram. Ensure the roach hides in darkness, runs from light, and responds to bumps when running from light (but not in darkness).
   - *Hint:* Initially test with Keyboard Input and TattleTale, then stimulate the roach hardware.
   - *Hint:* Create helper functions (e.g., `goForward(int x)`, `turnHalfSpeed(int x)`) to simplify your state machine code.
3. Get checkoff from a TA/Tutor.
4. For the lab report, include your updated FSM diagram, describe your strategy in designing and debugging the FSM, and detail any difficulties encountered. Include relevant code snippets and a **link to a video** of your working robot. You can use Google Drive with link sharing enabled.

Part 8 – Hierarchical State Machine

Overview:

Now that you have the basic functions down, let's make it more interesting by adding complex behaviors. The basic rules of the cockroach still apply: run from light, hide in darkness, and don't get stuck.

The new rules are:

- While hiding in the dark, if bumped, move away and continue for at least 0.5 seconds after releasing the bumper.
- While running in light, periodically (every 5-10 seconds), perform a dance or actively search for darkness. Be creative here.
- Ensure the "don't get stuck" behavior is robust to various scenarios.

You will use a Hierarchical State Machine (HSM) to implement this behavior. An HSM is like a flat state machine, but states can have their own sub-state machines. Events are passed to the lowest level state machine, and if handled, the event is consumed. If

[3] This is why we conditionally compile the test harnesses with a project-scope MACRO.

unhandled, it is passed up to the super-state machine. This allows for common reactions to be handled in one place, with specific reactions implemented as needed.

Reference Material:

- HSM Supplementary Lecture Video
- ElectricToothbrushFSM
- Tutorial: How to Start MPLAB X and ES_Framework
- ECE118 Roach Instructions
- Essential C – if you are unfamiliar or rusty with the C programming language
- CKO Chapters 1-5.

PreLab:

1. Spend time to discuss your HSM[4] 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.

Bumps and Road Hazards:

Put Roaches up on blocks when on table. Do **NOT** ever run your roach live on the lab benches. If they are moving, they go on the floor.

Tasks:

1. Set up a new ES_Framework project and use the HSM templates to create a simple hierarchical state machine.
2. Implement the HSM to achieve the three new rules specified in the Part 8 Overview:
   - While hiding in the dark, if bumped, move away and continue for at least 0.5 seconds after releasing the bumper.
   - While running in light, periodically (every 5-10 seconds), perform a dance or actively search for darkness.
   - Ensure the "don't get stuck" behavior is robust to various scenarios.
3. Get checkoff from a TA/Tutor.
4. For the lab report, include your updated HSM diagrams, describe your strategy in the design and debugging of your HSM, and detail any difficulties encountered. Include relevant code snippets and a link to a video of your working robot.

[4] In this case, there will be a top-level state machine (relatively simple) and each of those states will have a sub-state machine.

Checkoff and Time Tracking

Student Name:_____ CruzID_____@ucsc.edu

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

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