CSCI 1600 Final Project Report

1. Introduction

A short introduction giving an overview of your project and what assumptions you are making about the user.

Overview:

The automated Etch A Sketch we have built allows the user to translate drawings from a web interface to drawings on the Etch A Sketch. The user can draw on a canvas and click the “Start Drawing” button on the web interface which will then reset the Etch A Sketch cursor back to the origin and prompt the user to clear the screen should they want to do so. Once they confirm that they have done so and placed the Etch A Sketch on a flat surface, the drawing begins. The progress bar on the web interface and LEDs on the circuit allow the user to see how the drawing is shaping up in real time. The user can stop the drawing at any time by pressing the button on the circuit. They can then start another drawing from the web interface at any point.

Assumptions:

We assume that the user will have access to a computer that can run the web interface (React.js) and the web server (node.js) since the interface and server are currently not hosted anywhere and need to be manually run. We also assume the user is able to use an interface like ours that is just an MVP and has not been built or tested for accessibility. The user must also be able to read and understand English. The user must be able to pick up the Etch A Sketch and shake it with a reasonable amount of force to clear the screen and be capable of inserting 0.5mm diameter motor pegs into the shafts. Lastly, the user must be capable of pressing a small button on the breadboard that stops the system in the event of an emergency, or otherwise.

2. Requirements

Clear, concise, correct, coherent, complete, and confirmable requirements for your project, written with "shall/should" language as defined in class.

Parameters:

- The Etch A Sketch is navigated using coordinates where each coordinate corresponds to one step of the stepper motor. The coordinates of the top-left corner are (0, 0). The coordinates of the bottom-right corner are (330, 240).
- The Etch A Sketch is controlled from a web interface that communicates with a node.js server using web sockets and the node.js server communicates with the Arduino (that controls the Etch A Sketch) using Serial communication.

Requirements:

R0: The Etch A Sketch cursor *should* be manually placed at the top left corner of the Etch A Sketch by the user before running the software for the first time.

R1: The automated Etch A Sketch *should* account for slippage (from when the stepper motors are run too fast) and backlash (from the mechanical knobs of the Etch A Sketch) in the software.

R2: There are two stepper motors that *shall* be configured to control the left and right knobs of the Etch A Sketch and not force the cursor out of bounds of the screen.

R3: The web interface *shall* display a blank canvas on which the user can draw anything they like using their mouse. Once the user is done drawing, the user *should* click the "Start Drawing" button.

R4: Once the user clicks the "Start Drawing" button, the web interface *shall* instruct the Arduino to actuate the stepper motors to reset the cursor back to the top-left corner of the screen.

R5: The Arduino *shall* send a reset confirmation message upon completing the cursor reset process that *shall* be displayed on the web interface.

R6: Once the cursor reset process concludes, the web interface *shall* instruct the user to clear the screen by shaking the Etch A Sketch. The user *should* click the "Confirm" button once they have cleared the screen.

R7: The web interface *shall* then send the Arduino the first set of coordinates and the Arduino *shall* request the next set once it is done executing the prior, and does so until all coordinates have been plotted.

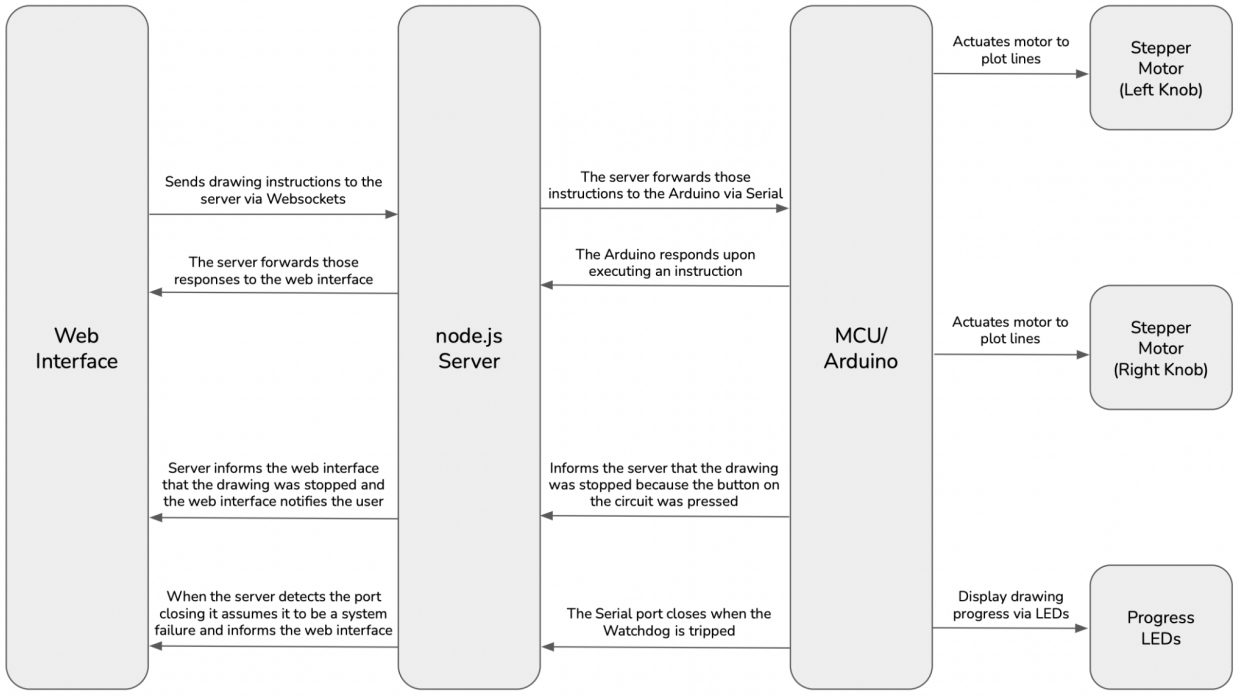
R8: The Arduino *shall* send progress updates on the drawing process that *shall* be displayed on the web interface via a progress bar and via an LED strip on the circuit.

R9: The circuit *shall* allow the user to stop the system completely by clicking the button which *shall* also reactivate all other interactive components on the web interface.

R10: The Arduino *shall* wait after completing the drawing process until it receives another command from the web server.

3. *Architecture Diagram*

An architecture diagram of the modules/components of the system and the interfaces between them. This should fit on one page and be a "boxes and arrows" diagram, as described in class.

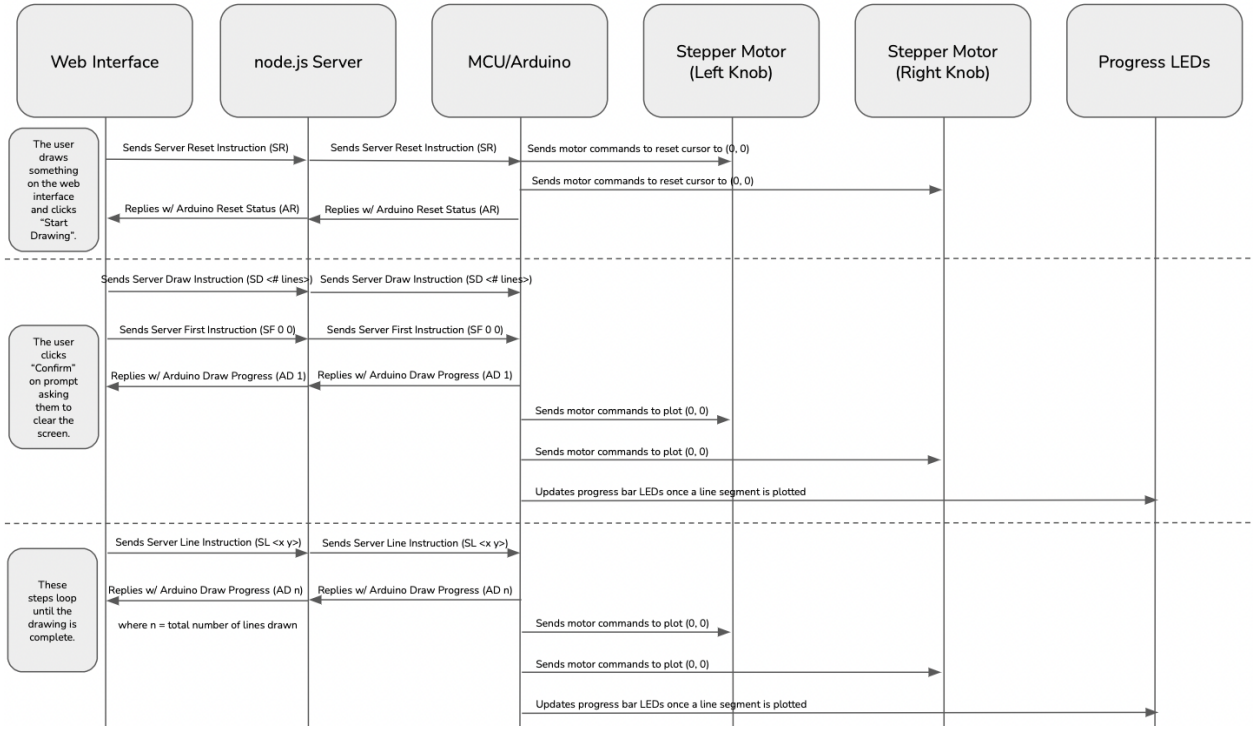


This is a link to a google slide presentation with the same diagram.

4. Sequence Diagram

A sequence diagram for each reasonable use case scenario of your system (if there are more than 4, you should pick the 4 most likely use case scenarios to diagram, and include a note as to what scenarios you left out). Before each diagram, you should write a short (1-2 sentence) description of what the use case scenario is.

Scenario 1: The user boots up the system for the first time, manually sets the cursor to the top-left corner and draws something on the web interface.



This is a link to a google slide presentation with the same diagram.

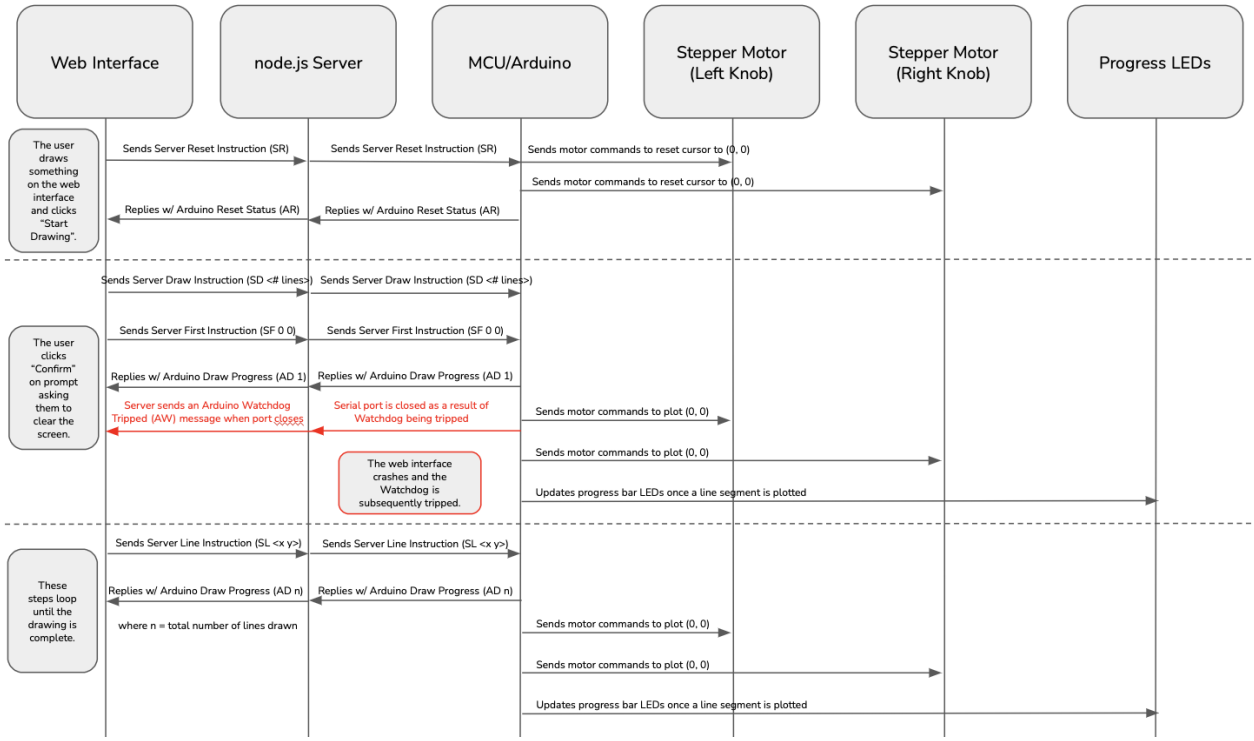
Scenario 2: The user boots up the system for the first time, manually sets the cursor to the top-left corner and draws something on the web interface. Once that drawing concludes, they decide to draw something else.

This is a link to a google slide presentation with the same diagram.

Scenario 3: The user boots up the system for the first time, manually sets the cursor to the top-left corner and draws something on the web interface. During that drawing, they decide they want to draw something else. So, they press the button on the circuit, which then stops the drawing immediately and they are informed that this operation was successful. They then proceed to draw something else.

This is a [link] to a google slide presentation with the same diagram.

Scenario 4: The user boots up the system for the first time, manually sets the cursor to the top-left corner and draws something on the web interface. During that drawing, the web interface is accidentally closed, or it crashes (or there is any kind of system failure), which causes line instructions to no longer be sent. This trips the Watchdog and shuts down the Arduino and the user is informed that there was a system failure, and that they must reset all systems manually before attempting to use them again.



This is a link to a google slide presentation with the same diagram.

5. FSM

One finite state machine that describes the detailed design of the main module of your system, as described in class. You should clearly define the inputs, outputs, and variables of the system. Remember to indicate how the variables are initialized and what the start state is. Also remember to number your states and give each one a concise name.

Inputs

The inputs are messages from the web interface received through the node.js server via Serial communication.

Outputs

The outputs are motor commands to sketch the drawing, messages to be sent to the web interface through the node.js server via Serial communication, and PWM values sent to the progress bar LEDs.

Variables

Name	Description
`lineInstructionsBuffer`	Circular buffer of length 255 in which `lineInstruction` structs are stored
`readPointer`	Read pointer for the `lineInstructionsBuffer`
`writePointer`	Write pointer for the `lineInstructionsBuffer`
`totalLinesToDraw`	Total number of lines to be drawn
`totalLinesProcessed`	Total number of lines that have been processed (i.e., parsed & stored in the buffer)
`cursorX`	x-coordinate of the cursor
`cursorY`	y-coordinate of the cursor
`latestX`	Most recent x-coordinate that was plotted
`latestY`	Most recent y-coordinate that was plotted

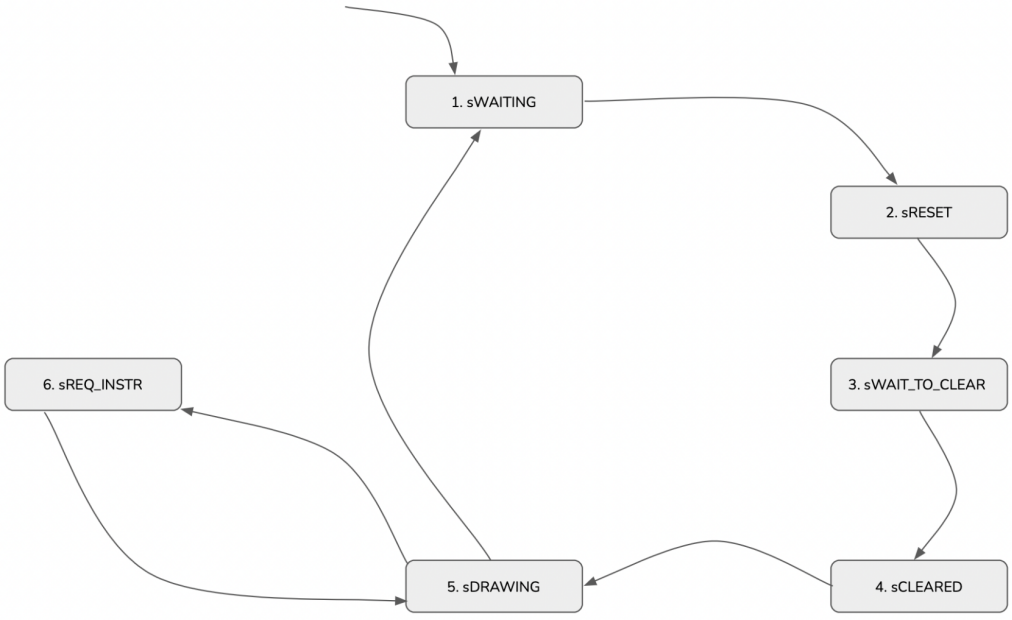
Variable Initialization

```
lineInstructionsBuffer = {0, 0}[255]
readPointer = 0
writePointer = 0
totalLinesToDraw = 0
totalLinesProcessed = 0
cursorX = 0
cursorY = 0
latestX = 0
latestY = 0
```

Functions

- `updateProgressBar()`: updates the luminosity of the progress bar LEDs using PWM. The first LED lights up with increasing luminosity up to the first 50% of drawing progress, after which the second LED starts lighting up. The LEDs flash 5 times after a drawing is complete. phaseSize is set to half the total number of lines to be drawn and incrementSize is set to 255 over phaseSize so that the increments to luminosity are uniform.
- `enableWatchdog()`: sets necessary bits in specific registers to enable the Watchdog. The Watchdog is only enabled when the Arduino starts receiving line instructions from the web server.
- `disableWatchdog()`: sets necessary bits in specific registers to disable the Watchdog. The Watchdog is disabled upon returning to the sWAITING state or upon the ISR being triggered.
- `resetSystem()`: resets all aspects of the system which includes resetting the cursor back to the origin (top-left corner), resetting all variables for the progress bar LEDs, and resetting all FSM variables.
- `plotLine(int x0, int y0, int x1, int y1)`: makes the appropriate call to plotLineLow(x0, y0, x1, y1) or plotLineHigh(x0, y0, x1, y1) depending on whether the slope is gradual or steep. It then updates the cursor's position to that of (x1, y1).

Diagram



This is a link to a google slide presentation with the same diagram.

Table

Transition	Guard	Explanation	Output	Variables
1-2	msg == “SR”	The “SR” (Server Reset Instruction) message informs the Arduino that the system must be reset as the user has initiated a new drawing.	resetSystem()	
2-3	cursorX == 0 ∧ cursorY == 0	Once the cursor returns to the origin, the Arduino sends the “AR” (Arduino Reset Status) message to the web interface. This signals the end of the reset process and now the user must clear the screen.	Serial.println(“AR “)	
3-4	msg == “SD <# lines>”	The user has finished clearing the screen, the web interface has converted the web sketch to coordinates, and sends the total number of lines to be plotted via an “SD” (Server Draw Instruction) message.	enableWatchdog()	totalLinesToDraw = <# lines>
4-5	msg == “SF 0 0”	The “SF” (Server First Instruction) message sends the first line instruction to be executed which will always just be to plot the origin. The Arduino responds with an “AD” (Arduino Draw Progress) message.	Serial.println(“AD <totalLinesProcessed>”)	add (0, 0) as a lineInstruction to lineInstructionsBuffer increment writePointer increment totalLinesProcessed
5-1	totalLinesProcessed == totalLinesToDraw	The drawing is complete and the Watchdog can be disabled.	disableWatchdog() updateProgressBar()	
5-6	cursorX == latestX ∧ cursorY == latestY ∧ totalLinesProcessed ¬= totalLinesToDraw	The prior line has been plotted and now the most recent lineInstruction (x, y) in the buffer will be plotted as long as there are elements left in the buffer.	plotLine(cursorX, cursorY, x, y) updateProgressBar()	latestX = x latestY = y increment readPointer
6-5	msg == “SL <x> <y>”	The “SL” (Server Line Instruction) message sends a line instruction to be executed which is parsed with extractLineInstruction(). The Arduino responds with an “AD” (Arduino Draw Progress) message.	Serial.println(“AD <totalLinesProcessed>“)	add (x, y) as a lineInstruction to lineInstructionsBuffer increment writePointer increment totalLinesProcessed

6. Traceability Matrix

A traceability matrix of how the transitions and states of the FSM in step 4 traces to the requirements in step 1. If your system is made up of multiple modules, you will not have full traceability from just one FSM, so you are not required to have a check in every column (requirement) of the matrix. Traceability matrices were covered in the FSM lecture.

States

	R0	***R1***	***R2***	***R3***	***R4***	***R5***	***R6***	***R7***	***R8***	***R9***	***R10***
1				X	X						X
2						X					
3							X				
4								X			
5		X	X					X	X		
6								X	X		

Transitions

	R0	***R1***	***R2***	***R3***	***R4***	***R5***	***R6***	***R7***	***R8***	***R9***	***R10***
1-2				X	X						
2-3						X					
3-4							X				
4-5								X			
5-1				X					X		X
5-6		X	X					X	X		
6-5								X	X		

R0 describes expected user behavior as a prerequisite for the system to function correctly and is hence not encoded in the FSM. **R9** describes the behavior of the ISR and is hence not encoded in the FSM.

7. Test Plan

An overview of your testing approach, including which modules you unit tested and which integration and system (software and user-facing/acceptance) tests you ran. You should explain how you determined how much testing was enough (i.e. what coverage or other criteria you used, how you measured that you achieved that criteria, and why you think it is sufficient for you to have met that criteria).

Unit Tests:

Unit tests were conducted for each state transition, including all non-transitions (i.e., where the guard to the transition is not met, and another call to updateFSM() is made with the same state with updated inputs). The metric we used for coverage was branch coverage. The branch coverage for the updateFSM() function was explained above in terms of state transitions, but other functions were covered by different test designs.

The functions for which it did not account for include enableWatchdog(), disableWatchdog(), interruptServiceRoutine(), and updateProgressBar(). All of these functions were tested in Integration or System tests in a way that guarantees branch coverage is met, i.e., one LED being lit in updateProgressBar() signifies the first branch being entered, two LEDs signifies the second branch being entered, and both LEDs blinking demonstrates that the third branch is entered. This behavior was observed in the system testing video linked below. The ISR functionality including all branches within interruptServiceRoutine() were tested via Integration Test Scenario 3; and the Watchdog functionality including all branches within enableWatchdog() and disableWatchdog() were tested via Integration Test Scenario 4.

All that remains, then, are the line plotting functions. The highest level function in this family, plotLine(), has a case for a line with greater change in x than in y, and vice versa, resulting in calls to plotLineLow() and plotLineHigh(), respectively; test 8 enters the first branch, and test 9 enters the second. In order to get branch coverage in each of plotLine()'s helper functions, when the change in x is greater than in y, dx must be positive (test 8) and then negative (test 10), and when the change in y is greater than in x, dy must be positive (test 11) and then negative (test 7).

Integration Tests:

An integration test is conducted for each of the four sequence diagrams (which covers the base cases of every possible permutation of events the user can initiate) where we test that every single message is sent and received as expected and also test that the sequence and timing of messages is correct. Unexpected messages are not tested because on both the web interface and server, as well as the Arduino, the only messages that we look for and parse are those in the specific format we've defined, anything else is just ignored.

System Tests:

For system testing, the Etch A Sketch was recorded performing drawings from the web interface and we visually compared what was drawn on the Etch A Sketch with what was drawn on the web interface, and found that an acceptable degree of recognition was present. Then, in order to test the ability of the system to work through multiple iterations, another drawing was entered and drawn over the existing drawing to verify that the system could work for arbitrarily many iterations. This is also present in the linked video.

8. Safety and Liveness

At least 2 safety and 3 liveness requirements, written in propositional or linear temporal logic, for your FSM. Include a description in prose of what the requirement represents. Each requirement should be a single logic statement, made up of propositional and/or linear temporal logic operators, that references only the inputs, outputs, variables, and states defined in part 5 of this report.

Safety

- G ¬(cursorX < 0 \ / cursorY < 0 \ / cursorX > 330 \ / cursorY > 240)
    - This requires that the cursorX and cursorY values stay within the boundaries, that is they are never outside the range from (0, 0) to (330, 240). We do this to ensure that the motors do not overextend the knobs in a particular direction.
- G ¬(totalLinesProcessed > totalLinesToDraw)
    - This requires that the value of totalLinesProcessed never exceeds the value of totalLinesToDraw. This is because we should only process line instructions after we receive them and it should only eventually equal the latter value; otherwise, it likely indicates a parsing error.

Liveness

- msg = “SR” → F (cursorX = 0 / \ cursorY = 0)
    - This states that if the msg equals “SR”, then, eventually cursorX and cursorY are equal to 0. Resetting the system should send the cursor to (0, 0) as this is used as the starting point for any drawing.
- F (totalLinesProcessed = totalLinesToDraw)
    - This states that it will eventually be the case that totalLinesProcessed equals totalLinesToDraw, indicating that the drawing is complete. This makes sense as we should never leave the system hanging or in an incomplete state.
- F (readPointer = writePointer)
    - This states that it will eventually be the case that readPointer equals writePointer. This indicates that all instructions in the buffer have been executed.

9. Modeling

A discussion of what environment process(es) you would have to model so that you could compose your FSM with the models of the environment to create a closed system for analysis. For each environmental process, you should explain and justify whether it makes sense to model it as either a discrete or hybrid system and either a deterministic or non-deterministic system (for example, a button push could be modeled as a discrete, non-deterministic signal because it is an event that could happen at an arbitrary time, but a component's position could be modeled as a hybrid, deterministic signal based on some acceleration command).

We would have to model communication between the Arduino and web server. Our current system acts in response to a user's drawing on the canvas and sends the corresponding instructions. We would model it as a discrete system because there would only ever be a finite number of instructions and instructions are sent one by one, in response to each other. If we wanted to model it as a deterministic system, we would need to use the same reference drawing for all runs. In this way, the Arduino will be interpreting the same set of commands every time.

We would also have to model the push button that triggers the ISR. Since this is an external hardware component, it would need to be composed to create a closed system for analysis. We would model it as a discrete system because there are only two values a push button can take. We would also model it as a non-deterministic system, since the intention of the ISR is that it can be triggered at any time and the system will stop the drawing in response. It can be implemented as another feature on the web interface. For example, a user hitting the "Stop Drawing" button on the web interface will trigger the ISR instead. We could add a flag in the Arduino code that ensures the ISR was triggered and transitions to the right state.

(We do not believe we would have to model the LED progress bar indicators, as the progress bar on the web interface acts as a suitable substitute.)

10. Code Summary

A description of the files you turned in for the code deliverable. Each file should have a high-level (1-3 sentence summary) description. ***For each of the assignment requirements (PWM, interrupts, serial, etc), you should indicate which file(s) and line(s) of code fulfill that requirement.***

Code Structure

- fsm/ (Contains the Arduino code)
    - fsm.h
        - Defines structs and enums, declares variables, and headers for functions.
    - fsm.ino
        - Defines the actual FSM; updateFSM() is responsible for transitioning states.
    - fsm_utils.ino
        - Defines the helper functions used by the FSM.
    - fsm_testing.ino
        - Defines the testing functions.
- interface/ (Contains the client-server architecture)
    - react-client/ (Most salient files from the react-client folder)
        - src/App.js
            - Sets up WebSocket connection and contains a function for parsing data from the server.
        - src/EtchPanel.js
            - Canvas seen on the web interface. Creates line instructions from collected canvas points.
    - server/ (Most salient file from the server folder)
        - server.js
            - Sets up Serial and WebSocket communication. The node.js server acts as the intermediary between the Arduino and the client.
- vectorize/ (Code for Abhinav's capstone feature)

Requirements

- **PWM** - **updateProgressBar()** in **fsm_utils.ino (lines 33 - 63)**
- **Watchdog** - **enableWatchdog()** and **disableWatchdog()** in **fsm_utils.ino (lines 79 - 111)**
- **ISR** - **initializeSystem()** and **interruptServiceRoutine()** in **fsm_utils.ino (lines 11, 65 - 76)**
- **Serial** - **updateFSM()** in **fsm.ino (lines 34 - 125** among other places in the codebase**)**

11. Running Tests

A procedure on how to run your unit tests (for example, if you have mock functions that are used by setting a macro, similar to lab 6, make sure to note that).

Running Unit Tests:
Comment out all of the code within the loop function in fsm.ino and uncomment the #define UNIT_TESTING macro in fsm.h. There are two functions that are called within the default implementation (fsm.ino) that are mocked out and enabled upon uncommenting #define UNIT_TESTING in fsm.h. The first is test_all_unit_tests() within setup() and the second is testing_state_edit() within updateFSM(). If you upload the files to any Arduino and open up the Serial monitor, you should eventually see “All unit tests passed!” printed out to Serial.

Running Integration Tests:
Comment out all of the code within the loop function in fsm.ino and uncomment the #define INTEGRATION_TESTING macro in fsm.h. There is one function that is called within the default implementation (fsm.ino) that is mocked out and enabled upon uncommenting #define INTEGRATION_TESTING in fsm.h, and it is test_all_integration_tests(). Running the integration tests is slightly more complicated than running the unit tests.

First, run `npm start` within /interface/react-client. Then, you will need to upload the files to any Arduino, and within 10 seconds of doing so, run the server file by running `node server.js --port <port on which Arduino is connected>` within /interface/server and also refresh the web interface to make sure you see the “SERVER UPDATE: Client Connection Established.” message in the terminal window running the server file before seeing the “INTEGRATION TESTING UPDATE: Initiating Integration Tests'' message.

If you do not see these messages in this exact sequence, perform all the tasks again, from the beginning. It could be helpful to run `node server.js --list` to find which ports are being used on your machine. You should eventually see “All integration tests passed!” printed out to the terminal window running the server file (do note that since the last integration test checks if the Watchdog functions as intended, you will need to manually restart the system after performing the integration tests should you want to continue using the system for other tests or to draw stuff on the Etch A Sketch - we are aware this is somewhat contrary to the behavior of a Watchdog but will explain why we do this in our individual reports).

12. Reflection

A reflection on whether your goals were met and what challenges you encountered to meet these goals. You should only include challenges met or newly addressed since the milestone.

Although the drawings on the Etch A Sketch were not perfect replicas of the drawings on the canvas, we were still satisfied with the results. It seems that the drawings on the Etch A Sketch would get the general structure right, but backlash/timing issues made it such that drawings would almost always be slightly distorted, so we would definitely need some fine-tuning there to counteract the various kinds of distortion factors we observed with different kinds of drawings.

Although we did fulfill the crux of the project, we did scale back the project ever so slightly. We changed projects at the milestone, and thus only had materials for our new project idea starting from the milestone date, so we did not get to work with them before we determined our goals. For example, we thought we could use a stepper motor to tilt a constructed frame for the Etch A Sketch to clear the image. This did not work as we realized you have to shake the Etch A Sketch relatively hard to clear the image and the stepper motors we ordered seemed like they were not nearly as powerful as they needed to be, to complete such a task. We also wanted to integrate a “Stop Drawing” button on the web interface, but we had trouble issuing an interrupt from the software, which is why we opted for a hardware interrupt.

Despite these challenges, we still managed to get the Etch A Sketch to draw. The biggest challenges we faced was communication between the Arduino and the web interface via the server, which proved to be difficult and was probably the part of the project we spent the most time on but we were glad we were able to get over this hump and demo the Etch A Sketch drawing as a cohesive embedded system that worked off a web interface.

13. Appendix

As an appendix, include the review spreadsheets you received after the milestone demo. Each defect should be marked as "fixed" or "will not fix" with a justification.

Project name:	**Etch-a-sketch**	
Defect record		
Transition or state #	**Defect**	**Checklist item**
State 1	Seems like it should be the initial state, but there is no initialization arrow	1
General	The inputs, outputs, and variables aren't defined. They can be inferred from the FSM table, but they aren't explicitly listed.	4
General	Input and variable initialisation is not included. Some can be inferred (e.g. *sketch* probably has an initial value of {}, but we can't know the initial values of variables like cursor_x or cursor_y).	5
Transition 1-2 and 1-3	Transitions between states are not all mutually exclusive. For example, in states 1-2 and 1-3, the guards are *sketch != {}* and *shake_received = true* - it is possible for both of these conditions to be true, which violates mutual exclusivity.	6
General	The FSM doesn't seem to address Arduino interactions with the web interface and doesn't cover converting uploaded images into line drawings	9

Project name:	Etch a Sketch
Defect record	
Transition or state #	**Defect**
State 1	Missing variable initializations
Transitions 1-2 and 1-3	Guards don't appear to be mutually exclusive -- what if the sketch is nonempty and a shake is received?

x	Start state is indicated
x	All states are numbered with a unique number
x	All states have a unique, short, descriptive name
	Inputs, outputs, and variables are defined
	Input and variable initialization is included
x	All transitions out of a state are mutually exclusive
x	Only inputs and variables are checked in guard conditions
x	Only outputs and variables are set in actions
x	FSM behavior aligns with behavior/requirements put forth in team presentation

The defects fell into the following categories:

Defect	Status
No initialization arrow for the initial state in the FSM diagram.	Fixed.
Inputs, outputs and variables are not defined.	Fixed (we just forgot to include this in our submission to the peer review gradescope drop).
Input and variable initialization not included.	Fixed (we just forgot to include this in our submission to the peer review gradescope drop).
Transitions are not mutually exclusive.	Fixed (we redesigned our FSM and now only one state, sDRAWING, has multiple transitions out of it, and those transitions are mutually exclusive).
FSM does not address interactions with the web interface.	Fixed (we did not know what our communication protocol was going to be at the time).
FSM does not address converting uploaded images into line drawings.	Will not fix. (since this is Abhinav's capstone feature, we decided to exclude it from the default submission).