22COA202 Coursework

F213619

Semester 2

1 FSMs

- *Describe the finite state machine at the centre of your implementation. Show the states and the transitions. Draw the states and transitions as a picture and include it here.*

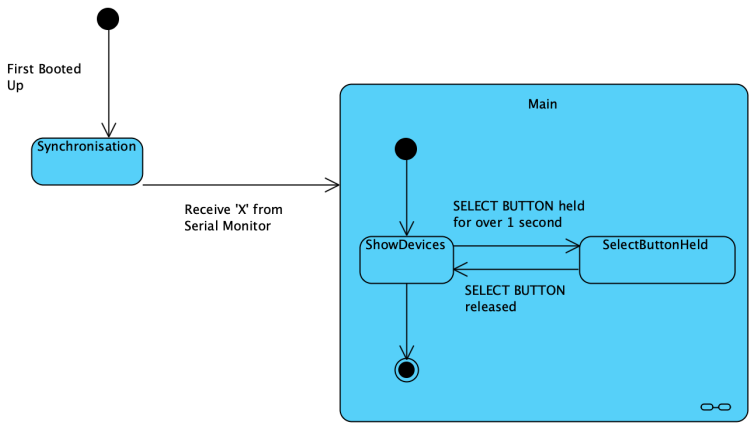
The implementation of my project involves two core states : Synchronisation (default when launched), and Main, designed to be executed within the main loop function of the program.

Within the Synchronisation state, the program sets the LCD BackLight to purple and prints 'Q' to the Serial Monitor at a frequency of once per second whilst waiting for the user to input a message into it. Once it has received a message, the program verifies that the input does not end with a New Line or Carriage Return and finally checks if the input is equal to 'X' - if so, the LCD BackLight is changed back to white and the string 'UDCHARS FREERAM' is sent to the Serial Monitor before transitioning to the Main state.

Within the Main state there are two sub-states referring to the activities of the Serial Monitor, and two sub-states referring to the LCD Screen's activity. Regardless of which state the Serial monitor is in, the ButtonHandler function is called and monitors button inputs.

- The LCD Screen states are : ShowDevices (default when launched) and SelectButtonHeld. Within ShowDevices, the program calls the showCurrentDevice function and displays the current device on the LCD Screen. If the Select Button is pressed whilst in this state, the program waits one second before transitioning to the SelectButtonHeld state. Once in the SelectButtonHeld state, the program transitions immediately back to the ShowDevices state if the Select Button has been released, otherwise it displays the F number and available SRAM until the button is released.

- The Serial Monitor states are : Processing (default when launched) and Processed. Within Processing, the program checks if the user has entered a message into the Serial Monitor, if so then the MessageHandler function is called, processes the message and returns the corresponding message to the Serial Monitor before transitioning to the Processed state. The Processed state checks that a device has been added and that the LCD is not in the SelectButtonHeld state, if so then the program calls the showCurrentDevice function and displays the device corresponding to the pointer the user is set to, before transitioning back to the Processing state.



2 Data structures

- *Describe the data structures you are using to implement the Coursework. These could be types (structures, enums), classes and constants. Also describe the variables that are instances of these classes or types.*
- *When you have functions to update the global data structures/store, list these with a sentence description of what each one does.*

In order to implement the Coursework, I chose to create a Device class with the following attributes (with their data type in brackets) : type (*deviceType*), state (*deviceState*), ID (*String*), Location (*String*), Power (*Int*), Temperature (*Int*). *deviceType and deviceState* are custom enums created to reduce errors in the creation of Device objects. The Power and Temperature attributes exist regardless of the object's type as I was unable to implement subclasses into my main deviceArray data structure to account for different types of Devices, therefore these attributes are set to 0 as default and not displayed for objects that 'should not have them'.

The deviceArray variable is an array of fixed length containing (10) Device objects. A given Device object is accessed using the global Pointer variable (Int) set to 0 as default, which represents the device's position in the deviceArray. When launched, the program initialises each of these device's attributes, and their ID is set to the empty string : "". This is used to identify 'unused' objects.

The alphabetical_sort function uses a bubble sort algorithm to sort the devices in deviceArray in alphabetical order by ID. It has been implemented in such a way that devices with empty strings as ID's (unused objects) are sorted to the back of the array.

The addDevice function receives the following parameters : ID (*String*), Location (*String*), and type (*char*), creates a new object using the given values and adds it to the deviceArray if there is enough space, before sorting it using the alphabetical_sort function. If it is unable to add the device to the deviceArray it will return a false boolean value, otherwise it returns true (this helps with returning a custom error / validation message to the user when executed). If called with a pre-existing device ID, the function will replace that device's attributes with the new ones provided.

The stateChange function receives the following parameters : ID (*String*), stateStr (*String*), and cycles through the deviceArray, changing the state of the device with the given ID to the new state then returns a validation or error message.

The pow_tempChange function receives the following parameters : ID (*String*), pow_temp (*int*), and cycles through the deviceArray, changing the power/temp of the device with the given ID to the new pow/state, then returns a validation or error message.

The removeDevice takes a device ID as input and cycles through the deviceArray, "deleting the corresponding device by setting its ID value to the empty string and then returns a validation or error message.

3 Debugging

Although I did not use any specific code to debug my program, my main process involved the Serial Monitor and printing variables throughout my code in order to track unwanted changes and errors wherever they might be.

4 Reflection

- *200–300 words of reflection on your code. Include those things that do not work as well as you would like and how you would fix them.*

Overall I am very pleased with how my code turned out, as I struggled implementing most features but eventually overcame every issue I faced. The main problem I encountered was the limited RAM which forced me to make certain decisions about the datatypes used, particularly in the definition of the Device class and where to use String variables.

This was a constant issue throughout the development of the project, and proved challenging when devices within the deviceArray would "lose" their attributes as RAM storage became scarce, and therefore print incomplete descriptions to the LCD display.

Something I found particularly useful was the F macro which freed up a lot of valuable SRAM space whenever the program would print any message to the Serial Monitor or LCD screen.

An odd bug that I confronted was the degree symbol not printing correctly on the LCD screen, meaning I would have to display a custom char whenever the LCD screen was set to a thermostat device.

While I'm happy with the final product, I do have an idea for improvement. I thought about creating two subclasses to the Device class, which would reduce the "wasted" or "unused" space in the deviceArray. Unfortunately, I wasn't able to implement this as I'd hoped.

5 UDCHARS

For this extension I created two custom characters : upwardsArrow and downwardsArrow, using two arrays of 8 bytes defined as global variables, initialised them in setup using the createChar function, and outputted them to the LCD screen using the lcd.write functions.

Overall I am pleased as to how they turned out!

6 FREERAM

For this extension, I was heavily inspired by the code sent via email which enabled me to find the free ram in my Arduino. From this, I tweaked the source code so that it would display the free ram on the Arduino's LCD screen once the function had been called.