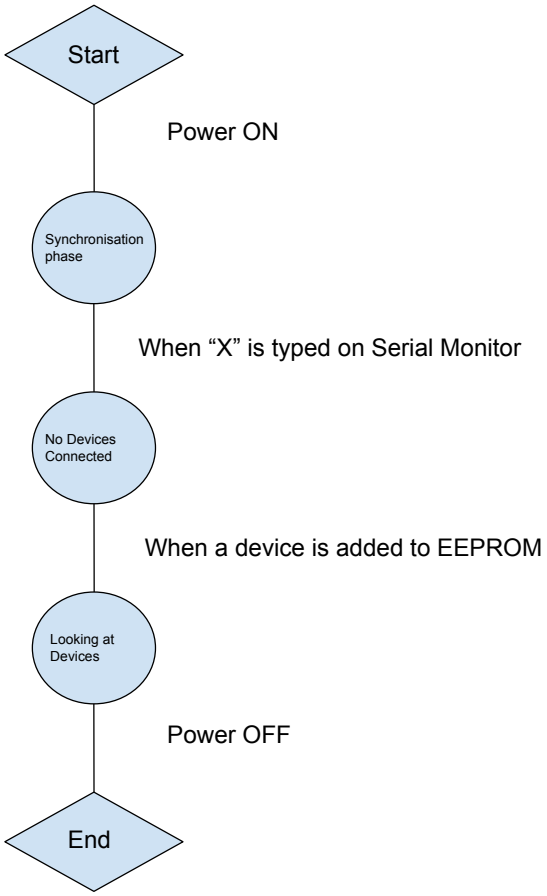
22COA202 Coursework

F214078

Semester 2 / 11/05/23

1 FSMs

The different states that I have in my code is the following. The first state is the synchronisation state, this is when the LCD is outputting a "Q" every second and the screen is purple. This then transitions into the next state when an "X" is inputted into the serial monitor. This state is when there are no devices connected or saved into the EEPROM and when there is a device that is added using one the protocols. This transitions into looping through all the different devices connected which can be changed by clicking the up or down button.



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.*

  The structure that I used to store the types of different data I needed and so that I could reuses the different variables inside the structure throughut my whole code is the following.

```
struct Device {
  String id;
  String location;
  byte power;
  String type;
  String state;
  String change_state;
  byte change_power;
};
```

The use of the data type string is self-explanatory, but the reason I chose to use byte instead of integers for the power is because when I tried to output the integers that were stored in EEPROM in the serial monitor or the LCD, it would corrupt and would output some random number instead of the correct number, so if I used byte it would store and read from the EEPROM correctly.

3 Debugging

```
void seeAllfunctions(int numDevices) {
  for (int i1 = 0; i1 < numDevices+1; i1++) {
    int startAddress = 24 * i1 + 1;
    Device device;
    // read id
    char buffer[4];
    for (int i = 0; i < 3; i++) {
      buffer[i] = EEPROM.read(startAddress + i);
    }
    buffer[3] = 0;
    device.id = String(buffer);
    // read location
    int i = 0;
    while (true) {
      char c = EEPROM.read(startAddress + 4 + i);
      if (c == 0 || i >= 15) {
        break;
      }
      buffer[i] = c;
      i++;
    }
    buffer[i] = 0;
    device.location = String(buffer);
    // read type
    char typeChar = EEPROM.read(startAddress + 19);
    device.type = String(typeChar);
    // read state
    i = 0;
    while (true) {
      char c = EEPROM.read(startAddress + 20 + i);
      if (c == 0 || i >= 3) {
        break;
      }
      buffer[i] = c;
      i++;
    }
    buffer[i] = 0;
    device.state = String(buffer);
    // read power
    EEPROM.get(startAddress + 23, device.power);

    // print device info
    if(device.id != NULL || device.id != ""){
      Serial.print(" ID: ");
      Serial.print(device.id);
```

```
      Serial.print(", Location: ");
      Serial.print(device.location);
      Serial.print(", Type: ");
      Serial.print(device.type);
      Serial.print(", State: ");
      Serial.print(device.state);
      Serial.print(", Power: ");
      Serial.println(device.power);
    }
  }
}
```

This code printed the devices in the Serial Monitor which helped see it easier when I was struggling with the LC.

4 Reflection

Limitations and possible fixes

- When I display a device it only dhows it for one second, I could make the delay longer by changing it to delay (5000) but then it will make it long if the user wants to flick through the devices quickly. So instead I could make it so that the device stays there until the next button is pressed. I could fix this possibly by making its own state and the next device would be its own state.
- **Please read EEPROM section to understand this part.** Another issue I have is connected to the way I have used EEPROM, the issue is that every time the power of the Arduino device is disconnected all the devices that were store in the EEPROM get deleted because of the way I use the counter. The way I could possible fix this is instead of starting from 0 every time the program, I carry keep the counter as it is and when I add a new device, I add that counter by one. So, say I have 4 devices already connected and the Arduino disconnects, then the next time I run it, it will add the new device at position 5.
- Right now, the device does not do anything when I click the left button if though the right button goes through the devices that are switched on, it could do the same but only show devices that are off.

5 UDCHARS

I used UDCHARS because I could add additional symbols or icons to my LCD display beyond the standard set of characters provided by the library. This was helpful in improving the style of my up and down arrows on the LCD screen when displaying the devices to let the user know whether there is another device if they press the up or down arrow. As this feature allows me to create my own custom characters by defining an array of 8 bytes representing the pattern of the character I want to create. This is what I did when I initialized it at the start of my code, shown by the following.

```
byte downArrow[8] = {
  B00100,
  B00100,
  B00100,
  B00100,
  B10101,
  B01110,
  B00100,
  B00000
};

byte upArrow[8] = {
  B00100,
  B01110,
  B10101,
  B00100,
  B00100,
  B00100,
  B00100,
  B00000
};
```

Then in my setup I had to include:

```
lcd.createChar(0, downArrow);
lcd.createChar(1, upArrow);
```

And I could then call it whenever I wanted with the following.

```
lcd.write(byte(0));
lcd.write(byte(1));
```

8 EEPROM

The way I have implemented reading and writing into my code is every time I run the code I set the very first address of the EEPROM to be 0 and this serves to be as a counter, and every time I add a device that counter increases by one.

The data that is stored in the EEPROM for one device is the following (device.id, device.type, device.state, device.power and device.location), the way I store these values into the EEPROM is:

- I store the 3 letters of device.id in addresses [1][2][3]

- The n number of letters for location get stored in addresses [4-18] with a maximum of 15 letters and if it less than 15 then it will just get padded with null values.

- The letter for the type gets stored in address [19]

- The 3 letters of device.state in addresses [20][21][22] and if is "ON" then it just pads 21 as null.

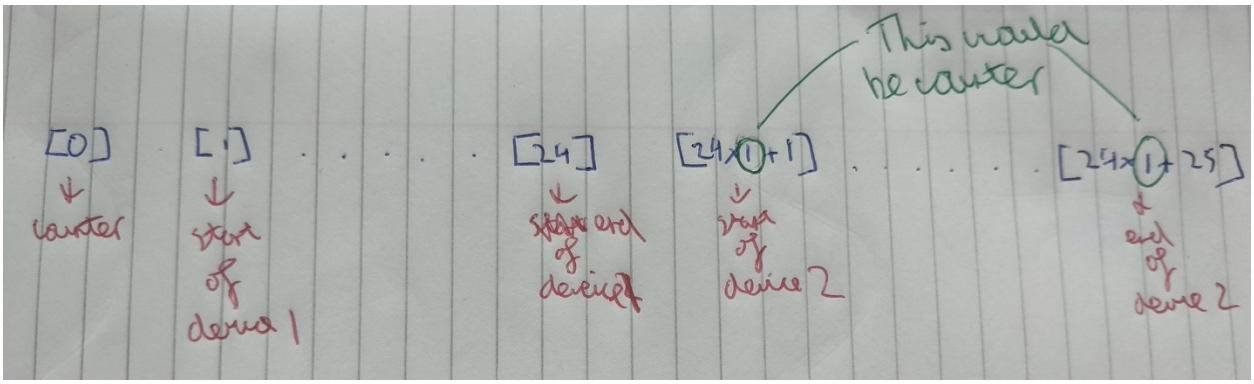
-The byte of power gets stored in the address [23].

A picture of the structure of this can be seen in the picture below.

The way I store multiple devices is I use the counter so say I am adding a new device and there are already 2 devices then the start address for the device where the device.id will start will be.

```
int startAddress = 24 * counter + 1;
```

The reason I multiply by 24 is because each device contains 24 address then I multiply by the counter device and I add one because the first address of the EEPROM, address[0] is the counter.



For reading a device, if I am looking for a specifc device.id, I will then just loop through the amount of devices there, every 24 addresses in the EEPROM, until I find the device.id I am looking for.