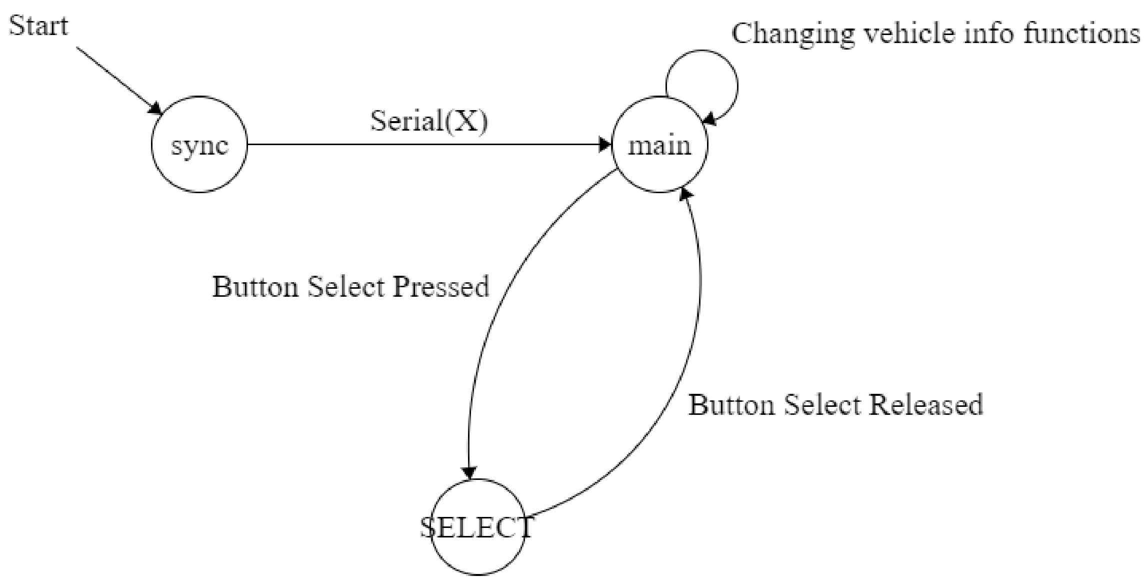
23COA202 Coursework

F319748

Semester 1

1 FSMs



The initial state is the sync state. The only way to get from the sync state to the main state is to input “X” into the serial monitor. Anything else that is inputted, does not change the state and comes up with error messages. Once “X” has been inputted into the serial monitor, then we transition into the main state. The main state relies on inputs from the serial monitor or the buttons on the Arduino. If a valid input from the serial monitor is detected, the main state calls on functions to execute the desired actions relating to the vehicles. Once these actions are complete, the functions then set the state to main once again causing a self-loop within the FSM. If the select button on the Arduino is pressed and held for one second or more, the state then changes to the SELECT state. Once in the SELECT state, then the student ID number and the amount of free RAM available is shown. Once the select button has been released, then the state transitions from SELECT to back to main.

2 Data structures

A struct “VehicleInfo” has been used to keep the information for each vehicle added which includes vehicle registration number, vehicle type, vehicle payment status, vehicle parking location, enter parking time and paid parking time.

Byte arrays have been used to store the customised up and down arrows.

A character array has been used to split the string vehicle parking location into separate characters to implement the SCROLL extension.

A class that has been used is the “Adafruit_RGBLCDShield” which allows for interaction with the LCD screen.

The constants in my program are the states which are “sync”, “main” and “SELECT”.

The key global variables are: “state” which shows which state the FSM is currently in; “currentTime” which uses millis() to keep track of the time in milliseconds from when the program has started to run; “displayedVehicle” which keeps track of the position of the vehicle in the array that needs to be displayed and finally multiple Boolean variables like “selectPressed” and “display” to make sure the buttons and display work well together.

The function “AddToList” adds the vehicle inputted by the user into the structure as long as it is valid, and the vehicle is not already there. If it is, then the vehicle type and parking location can be changed.

The function “ChangePaymentStatus” changes the vehicle payment status from “NPD” to “PD” as well as adding in the paid parking time. It also changes the status from “PD” to “NPD” which then causes the paid parking time to be erased and the enter time to be updated.

The functions “ChangeVehicleType” and “ChangeParkingLocation” both check inputs are valid before updating the array for the correct vehicle.

The function “RemovingVehicle” checks if the vehicle exists and the parking has been paid before removing the vehicle by shuffling each vehicle, after the vehicle to be removed, up by one position.

The function “displayVehicleInfo” simply displays all the information to the Arduino and also implements the SCROLL extension.

The function “buttons” controls what the up, down and select button do, by incrementing and decrementing the displayedVehicle variable.

3 Debugging

I have used multiple debug statements throughout my program. These have allowed me to discover errors within my code and correct them. For example, in the CheckParkingLocation() function, I had to include a debug statement to make sure that the code was working as it should. Initially for the if statement within that function, I had forgotten to check if the parking location length was greater than zero, otherwise blank parking locations would have been allowed in the program which would be incorrect. Debugging allowed me to spot my error and correct my code so now only valid parking locations between 1 and 11 characters is allowed. As well as this, testing certain parts of my code by simply printing out different variables that were being used and saved helped to debug my code further. An example of this is within the AddToList() function in my program. Printing out what was going to be saved in the array allowed me to check for formatting mistakes within my variables. This aided me in formatting the time correctly for the enter parking time where I had to include leading zeroes for single digit numbers.

4 Reflection

I think that overall, I am quite happy with how my code turned out. In particular, I am proud of implementing the millis() function in order for the parts of the code that needed to count seconds to work correctly. I am also proud of implementing the SCROLL extension to my program as this proved to be quite difficult to figure out how to get the parking location to wrap around itself when displaying on the Arduino. As well as this, I am happy with how the time formatting came out for the enter parking time and the paid parking time as this took time to determine how to get the leading zeroes for the single digit numbers and concatenate the hours and minutes to one string as initially I had them as integers.

I think in terms of improvements to my code, displaying the vehicle information to the Arduino could have been done better. This is because sometimes, when displaying a vehicle, if some button is pressed or an input is given in the serial monitor for example, it does not always return to the vehicle that was previously displayed before that action was taken. I would fix this by implementing some kind of variable that keeps track of what vehicle is displayed, which can then be used to display the correct vehicle. As well as this, sometimes, if the up and down arrows are pressed too much, the parking location becomes a random jumble of symbols. I found that once the select button was pressed for one second and released, then the program would return to normal for a while, but the problem persists after this. To fix this, I would perhaps decrease the size of the array holding the vehicle information which should hopefully allow the program to function normally.

Extension Features

5 UDCHARS

This extension makes use of the functions written within the code: arrows(), displayVehicleInfo(), VehicleAfterIndex() and VehicleBeforeIndex(). The arrows() function is where the particular up and down arrows are created using byte arrays which is important so that it can control each pixel. It allows the customised arrows to be formatted properly with 5 columns and 8 rows of pixels. After the characters are created, they then need to be displayed according to what position a vehicle is being displayed on the Arduino. If there is only one vehicle in the VehicleInfo structure, then there are no vehicles before the particular vehicle in the structure as checked by VehicleBeforeIndex() and there are now vehicles after as checked by VehicleAfterIndex(). Therefore, no arrows will be displayed. Otherwise, if there is not a vehicle before the particular vehicle position – the vehicle is in position 0 in the struct – then only the down arrow will be displayed. Similarly, if there is not a vehicle after the particular vehicle position – the vehicle is the last vehicle in the struct – then only the up arrow will be displayed. Otherwise, both arrows will be displayed as there is both vehicles before and after the particular vehicle. The arrows() function is then called upon every time the displayVehicleInfo() function is called upon in order to adjust the arrows for each vehicle being displayed on the Arduino.

6 FREERAM

In order to implement the FREERAM extension, you first have to include the library “MemoryFree.h”. Once this has been included, while the select button is pressed, the program is now in the SELECT state and the amount of free memory left is displayed using the function “freeMemory” from the library along with my student ID number. Once the select button is released, the state returns to the main state and goes back to displaying the list of vehicles.

7 SCROLL

The scroll extension is implemented within the displayVehicleInfo() function. This all takes place within the main state. When displaying each vehicle, within displayVehicleInfo(), there is an if statement to check whether the vehicle parking location is greater than 7 characters. If it is not, then the program simply displays the parking location without scrolling as it fits perfectly onto the lcd screen. However, if the parking location is greater than 7 characters, then the location is saved as a string into the variable called "loc". Then, "loc" is converted into a character array with the name "locationScroll". Then a while loop is created so that the scrolling effect is implemented until either a button is pressed, or an instruction has been inputted into the serial monitor. A variable "position" is created to change the position of the characters that will be displayed. To make the characters scroll at 2 characters per second, an if statement is then created to find the difference between the previous time and the current time using millis(). The parking location is then printed onto the lcd screen letter by letter from the character array using the position variable and the length of the locationScroll array.