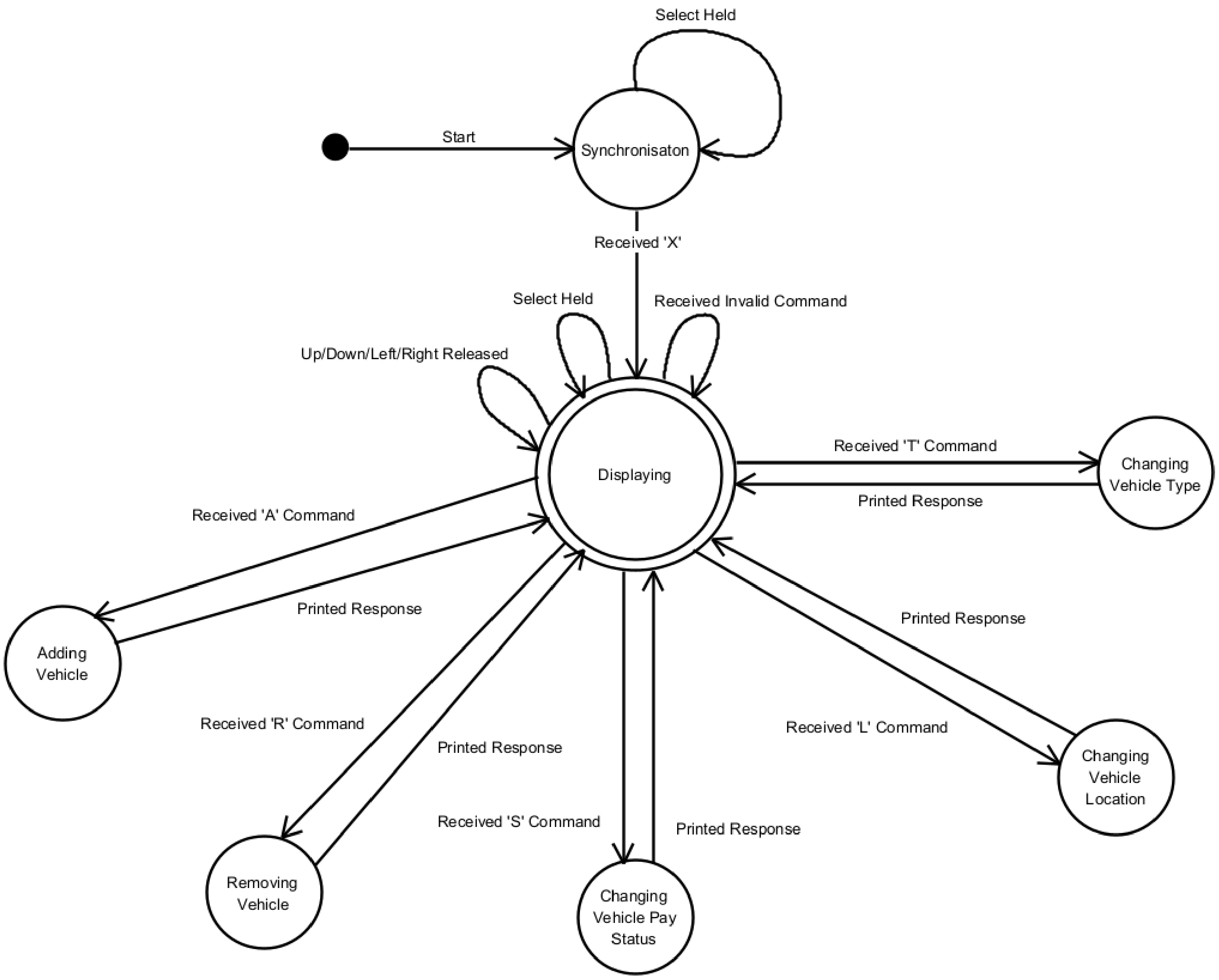
1 FSMs


Select Held
Start
Synchronisaton
Received 'X'
Select Held
Received Invalid Command
Up/Down/Left/Right Released
Displaying
Received 'T' Command
Changing Vehicle Type
Printed Response
Received 'A' Command
Printed Response
Adding Vehicle
Printed Response
Received 'R' Command
Printed Response
Received 'L' Command
Changing Vehicle Location
Received 'S' Command
Printed Response
Removing Vehicle
Changing Vehicle Pay Status

Synchronisation Phase

The state the program begins in. Prints ‘Q’ to the serial port every second until ‘X’ is received.

Transitions

Select Held: Changes display to purple and displays student ID and how much SRAM is currently free.

Received ‘X’: Synchronisation message is received, prints the extension list and sets backlight to white.

Displaying

The main state the program will occupy after synchronisation and the only accepting state. Updates the display based on button presses and listens for incoming serial messages.

Ensures that messages received are of a known format.

Transitions

Select Held: Shows the select menu as in the synchronisation phase.

Received Invalid Command: Receives a command that is not recognised or an invalid vehicle reg number.

Up/Down/Left/Right Pressed: Changes the currently displayed page in the list depending on the pressed button. Left and right filter by unpaid and paid respectively.

Received <X> Command: Received a recognised command with a valid vehicle reg number. Transitions to the state corresponding to the command, which will perform its own validation on the other command parameters.

Possible Errors:

Invalid command format

Invalid reg number

Command States

The remaining states all represent the servicing of one of the command types. Each of these states checks that the relevant number of parameters is specified and validates them before continuing the process. They print an error or 'DONE!' before returning to the displaying state.

Under each state I have described the validation performed by that state and listed the possible errors that can occur within that state.

Transitions:

Printed Response: Prints a relevant error if process fails, prints DONE! Otherwise.

Adding Vehicle

Attempts to add a vehicle to the list or update its information if it has paid.

Possible Errors:

Invalid vehicle type

Invalid location

Vehicle limit reached

Cannot modify unpaid vehicle

A vehicle with the same reg number, type and location already exists

Removing Vehicle

Attempts to remove a vehicle from the list.

Possible Errors:

Excess parameters specified for command

Vehicle not found

Cannot remove unpaid vehicle

Changing Vehicle Pay Status

Changes a vehicle’s pay status to the specified value.

If changing from paid to unpaid, treat the vehicle as a new entry.

Possible Errors:

Excess parameters specified for command

Vehicle not found

Vehicle already has this status

Invalid pay status given

Changing Vehicle Type

Changes a vehicle’s type.

Possible Errors:

Excess parameters specified for command

Invalid vehicle type

Vehicle not found

Cannot change type of unpaid vehicle

Vehicle is already of this type

Changing Vehicle Location

Changes a vehicle’s location if it has paid.

Treats the vehicle as a new entry.

Possible Errors:

Excess parameters specified for command

Invalid location

Vehicle not found

Vehicle is already in this location

Cannot change location of unpaid vehicle

2 Data structures

Vehicle Class

This class represents an entry in the list of vehicles.

All of the class attributes are public for the sake of convenience, since there is no way for a user's input to directly affect them without being vetted by validation functions first.

Throughout the program, a vehicle type of NULL is used to represent an empty vehicle.

Private

Methods:

CreateTime: This function formats the current time into the 'HHMM' format expected by the display and assigns it to a character array.

Public

Attributes:

- **char array** Registration number
- **char array** Entry timestamp
- **char array** Exit timestamp
- **char array** Location
- **char** Type
- **bool** Payment status
- **bool** From storage (used for EEPROM)

Methods:

CreateEntryTime: Creates an entry timestamp using CreateTime().

CreateExitTime: Creates an exit timestamp using CreateTime().

ClearExitTime: Resets an exit timestamp to whitespace.

Constructors:

There are two definitions for the constructor.

One takes no arguments and simply assigns the vehicle's type as NULL.

The other accepts arguments for reg number, type and location and creates a populated vehicle entry with the current time as entry time.

Updating Functions

ChangePayStatus: Changes a vehicle's pay status provided it does not already have this status.

ChangeType: Changes a vehicle's type provided it does not already have this type.

ChangeLocation: Changes a vehicle's location provided it has paid and is not already in this location.

Vehicle Array

The vehicle array vArr represents the list of all vehicles in the program.

Updating Functions

RemoveVehicle(): Remove a vehicle with a given reg number. This shifts all of the following array entries left by one in order to ensure that the entries are always sorted by entry time.

AddVehicle(): Add a vehicle with a given reg number, type and location.

FilterVehicleArray(): Filter the list of displayed vehicles by pay status.

UpdateDisplay(): Updates the display to show the relevant information for the current page in the vehicle array.

Command Array

The array cmdArr is a 2-dimensional character array used to store each section of a command.

Updating Functions

ReadMessage(): Read received command from the serial port and ensures its formatting is correct. Then, split the message into sections using dashes as a delimiter and write each section into the entries of cmdArr.

Other Structures and Variables

enum states: DISPLAYING, ADDING_VEHICLE, REMOVING_VEHICLE, CHANGING_PAY_STATUS, CHANGING_TYPE, CHANGING_LOCATION

instance state

Represents the current state within the main phase, includes all states except synchronisation.

char commands

List of all valid commands, used to validate commands from received messages.

char types

List of all valid types, used to validate vehicle types from received messages that require a type.

int vArrLen

The maximum length of the array that stores vehicles, i.e., the maximum number of vehicles that can be added to the list.

int index

The index in vArr of the current vehicle being displayed, in other words the page number.

3 Debugging

I didn’t realise we were supposed to comment out debug code, so I deleted it once I was finished using it to prevent printing messages that break the debugger. I have left in my debugging code that indicates a function has executed successfully.

4 Reflection

- I used the setup function as the synchronisation phase since the one-way transition to main phase lends itself to the setup and loop functions. However, this somewhat ruins the structure and I dislike how it turned out despite it working, so I would change this in the future.
- My UpdateDisplay function performs several discrete functions, so if I had more time I would decompose it to make my code more readable.
- Most of my ‘global’ variables are actually static variables in the loop function. This is due to my misunderstanding how static variables worked when beginning the project. This makes the code less readable and complicates my functions without providing any real benefits, and I would like to fix it in the future.
- I tried to use interrupts to implement the time-sensitive conditions, but this seemed awkward for any function where the interrupt would change variable values since reading those values would require disabling all interrupts.
- I tried to implement get/set functions for my vehicle attributes, but I couldn’t find a good method to return a character array from a function, so I scrapped this idea. I felt that encapsulation wasn’t too important for program safety since no user input can directly affect vehicle attributes without being validated.
- Pressing and releasing buttons quickly increases the odds of dropped inputs, in the future I would try to optimise my button handling code since I think it’s too slow to read a press and release in quick succession.
- There’s a lot of overlap between the processes in functions, e.g. some parts of input validation are done in the display phase and some when processing a command. I did this to avoid duplicating code, but it would be worth expanding to improve readability in the future.

5 UDCHARS

All I did for this extension was add byte arrays for the up and down arrow characters, initialise them as characters in the sync phase and then display them instead of the ^ and v characters on the display.

Functions and Variables

byte upArrowArr: byte array that holds the up arrow character.

byte downArrowArr: byte array that holds the down arrow character.

6 FREERAM

Relatively simple since the MemoryFree library allowed me to just get the amount of free SRAM with no hassle.

I used an interrupt to update the free memory every half second. This allows the select display to show the free memory in real time. Without the interrupt, the select menu would only show the amount of free memory at the moment when the display transitioned to the menu, which would not change until select was released and held again.

Functions and Variables

int freeMem: Volatile int used to hold the current free memory.

UpdateFreeMem(): Interrupt handler that sets freeMem to the current free memory.

7 HCI

In order to filter my array, I created a second array of the same size into which the filtered results are copied and displayed.

Unfortunately, this doubles the amount of memory reserved for each increment of the maximum vehicle size, but I couldn't think of any other way to do this that wouldn't result in overwriting some of the stored vehicles.

The silver lining for this is that this means I will never be able to store enough vehicles in SRAM to risk running out of EEPROM to store the vehicles to, since there is twice as much SRAM as EEPROM and some SRAM is already occupied by the program.

To some extent, the filters can be thought of as substates of the displaying state. However, all of the logic outside of the display is the same between the three, I didn't think they were distinct enough to actually classify them separately or using a sub-state diagram.

Functions and Variables

enum currentDisplay: MAIN, UNPAID, PAID

instance display: This enumeration is used in the DISPLAYING state to determine whether to display a filtered array or not.

vArrSub: A second vehicle array of the same length as the main array. When filtering the main array by pay status, all vehicles with the status to filter by are copied into this array, which is then used by the UpdateDisplay function instead of the main array.

FilterVehicleArray: Clears the sub array, then takes a given pay status and copies all entries from the main array with this status into the sub array.

8 EEPROM

My implementation reads from EEPROM expecting the following format:

- First byte is set to 255 to indicate data has previously been stored – this prevents reading from EEPROM if what is stored there is not from this program.
- Next 4 bytes store the timestamp of the last operation that updated stored information. This timestamp is not tied to the real time, it starts at 12:00:00 on 01/01/2023 and progresses as the arduino is left on.
- After this, each block of 33 bytes contains the information for each vehicle in the array (vehicle attributes occupy 33 bytes in total)

The data is read once the sync phase finishes, and a debug message is printed to indicate if data was read and what was retrieved (to demonstrate that the flag for stored vehicles works). Since messages can't be sent during the sync phase without producing unexpected results from the debugger, this will print AFTER the debugger writes its first message to the serial port.

Using the read and write commands correctly was very finnicky since each of the commands takes arguments in a slightly different format, for example I had an error at one point because I was trying to read data with an 8-bit address value, not realising that the address would overflow resulting in incorrect data being read.

There are some flaws in my implementation:

- If the maximum array size is decreased, the stored data for the vehicles which are now too far down the list to be retrieved is still kept in EEPROM, so when the array size is increased again these vehicles will still be present, which may not be desirable. This could be fixed by clearing all bytes beyond the final vehicle whenever the array is stored.
- If the first byte in EEPROM happens to be 255 before data has ever been written by this program, then the program will read memory anyway and likely corrupt. The odds of this could be reduced by increasing the number of bytes allocated for this check.
- Since both the add and remove commands involve potentially shifting multiple array values left, I had them both write the entire array to EEPROM when they are executed. This makes the AddVehicle and RemoveVehicle functions extremely slow compared to before implementing EEPROM, which increases response times and also means the display freezes during the execution of these functions. I could improve this by having only the shifted values in the array be written to memory, but this would still be very slow if removing an entry near the start of the array.

Functions and Variables

StoreVehicle(): Stores one vehicle to a block in EEPROM based on its position in the vehicle array.

StoreArray(): Stores the entire vehicle array to EEPROM.

ReadData(): Reads all stored data from EEPROM if the flag indicating presence of stored data is set.

9 SCROLL

I implemented scrolling by creating a static copy of the displayed vehicle's location and then shifting all of the characters in this copy left by one every half second, resetting it to the original location at the end. All scrolling logic is handled within the UpdateDisplay function.

I had to take extra care to ensure that the scroll reset to the beginning of the location every time the page changed.

As with other real-time display updates, the scrolling text freezes when a command writes the vehicle array to EEPROM.

One strange behaviour is that if the select menu is being displayed, the text keeps scrolling as expected unless the page is changed while the menu is displaying, in which case the location string will not scroll until select is released again. However, this doesn't cause any issues so I did not fix it, in the future I would try to figure out what causes this.

Functions and Variables

None