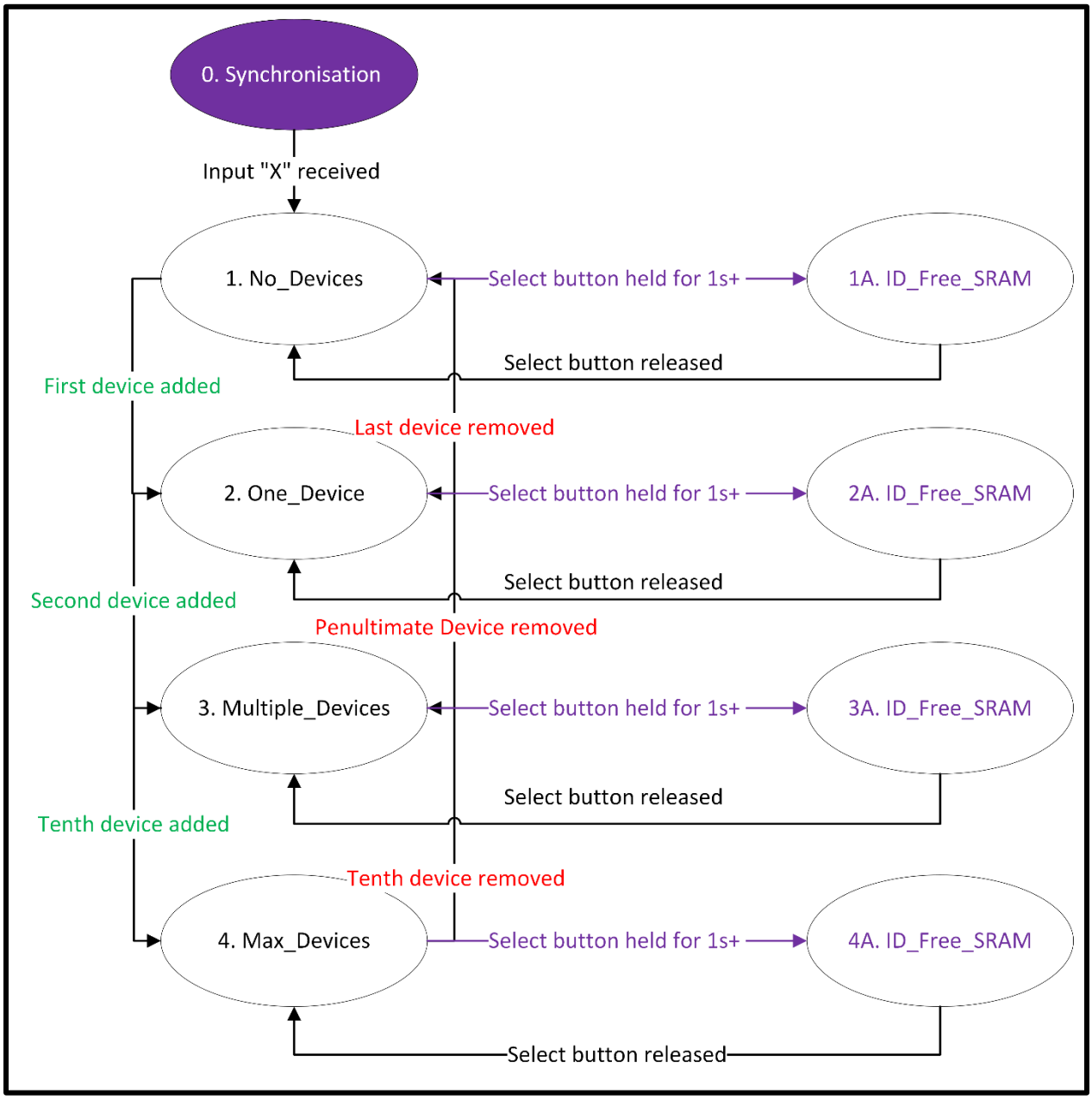
22COA202 Coursework

F221887

Semester 2

1 FSM


0. Synchronisation
Input "X" received
1. No_Devices
Select button held for 1s+
1A. ID_Free_SRAM
Select button released
First device added
Last device removed
2. One_Device
Select button held for 1s+
2A. ID_Free_SRAM
Select button released
Second device added
Penultimate Device removed
3. Multiple_Devices
Select button held for 1s+
3A. ID_Free_SRAM
Select button released
Tenth device added
Tenth device removed
4. Max_Devices
Select button held for 1s+
4A. ID_Free_SRAM
Select button released

0. Synchronisation

- Upon being booted up the home monitor enters this state, where the backlight is set to purple, and “Q” is sent to the serial monitor at a frequency of one per second.
- In order to enter the next state, an input of “X” must be received.
- When this happens, the backlight is set to white, and “BASIC\n” is sent to the serial monitor. The machine then enters state 1.
- Once this state has been exited, the machine cannot return to it until rebooted.

1. No_Devices

- This state is entered once the synchronisation phase is complete.
- The state is also entered upon deletion of the only device in the device array.
- During this state, the machine is listening to the serial monitor for inputs beginning with “A” which can then be validated to check if they conform to protocol.
- If the input is valid, the device is added to the device array and state 2 is entered.

2. One_Device

- This state is entered when the first device is added.
- It is also entered when enough devices have been deleted so that only one remains.
- The lcdDisplay() function is ran, which displays the device according to formatting requirements.
- During this state the machine is listening to the serial monitor for inputs beginning with any of:
    - “A” – add device
    - “S” – turn device on or off
    - “P” – adjust power setting of device
    - “R” – remove device
- If valid input for removing a device is detected, the device will be removed and state 1 will be entered.

- If valid input for adding a device is detected, the device will be added and state 3 will be entered.

3. Multiple_Devices

- This state is entered when a second device is added.
- It is also entered when the tenth device is removed
- The lcdDisplay function is ran, which displays the current device according to formatting requirements.
- During this state the machine is listening to the serial monitor for inputs beginning with any of:
    - "A" – add device
    - "S" – turn device on or off
    - "P" – adjust power setting of device
    - "R" – remove device
- The machine also listens for inputs from the lcd up and down keys, which act as a scroll function by updating the current device value to be used in the lcdDisplay function.
- If valid input for removing a device is detected and the device count is currently 2, the device will be removed and state 2 will be entered.
- If valid input for adding a device is detected and the device count is currently 9, the device will be added and state 4 will be entered.

4. Max_Devices

- This state is entered when a tenth device is added (the maximum number allowed).
- The lcdDisplay function is ran, which displays the current device according to formatting requirements.
- During this state the machine is listening to the serial monitor for inputs beginning with any of:
    - "S" – turn device on or off
    - "P" – adjust power setting of device
    - "R" – remove device

- The machine also listens for inputs from the lcd up and down keys, which act as a scroll function by updating the current device value to be used in the lcdDisplay function.
- If valid input for removing a device is detected, the device is removed and state 3 is entered.

1A, 2A, 3A, 4A. ID_Free_SRAM

- These 4 states act as substates of states 1-4.
- The state is entered by holding the select button for over 1 second.
- The backlight is set to purple, and my student ID number and number of free RAM is displayed.
- The states listen for the same serial monitor inputs as the state they were entered from, hence why they are 4 separate states rather than one.
- For instance, if entered from state 1 it will only listen for “A” inputs from the serial monitor.
- The state is exited when the select button is released, and returns to whichever state it was entered in.

2 Data structures

Structures and Variables

Struct Device
This structure is used to represent a device stored on the home monitor and is defined with 5 characteristics:

- String id – the unique ID for the device
- String location – the location for the device
- Char type – the letter representing the type of device – ‘O’ , ‘S’, ‘C’, ‘T’ or ‘L’
- Bool state – whether the device is currently on (true) or off (false)
- String data – represents the power output of the device (% or temperature)

device deviceList[10] – stores the list of devices, is used to add, remove and update devices

device sortedList[10] – stores the devices in alphabetical order, is used for displaying devices in the correct order

int deviceCount – stores the number of devices, is used to check devices can be added and for iterating through current devices more efficiently

int currentDevice – stores current device to be displayed, is updated when the scroll function is used

int SelectStartTime – is used to determine when select button has been held for a second.

bool isHoldingSelect – is used to determine when select button has been released.

Relevant Functions

void alphabetSort() – this function sets sortedList[] to be equal to deviceList[], then sorts sortedList[] by ID alphabetically. It is called at the start of the loop function to allow for the list to be in the correct order before being displayed.

void addDevice() – converts user input to device structure and adds to deviceList[], before incrementing deviceCount.

void setDevice() – converts user input to an ID and Boolean value, then updates the device.state of the device in deviceList[] with the given ID.

void dataDevice() - converts user input to an ID and a data value, then updates the device.data of the device in deviceList[] with the given ID.

void removeDeviceByID() – converts user input to an ID and sets all values in deviceList to the next value starting from that index, before decrementing deviceCount.

void selectPressed() – is used to create states 1A, 2A, 3A and 4A, achieves this by using the isHoldingSelect and SelectStartTime variables.

3 Debugging

```
/* void testDeviceList() {
  for (int i = 0; i < deviceCount; i++) {
    Serial.print("Device ");
    Serial.print(i);
    Serial.print(": ID=");
    Serial.print(deviceList[i].id);
    Serial.print(", Type=");
    Serial.print(deviceList[i].type);
    Serial.print(", Location=");
    Serial.print(deviceList[i].location);
    Serial.print(", State=");
    Serial.print(deviceList[i].state);
    Serial.print(", Data=");
    Serial.println(deviceList[i].data);
  }
} */
```

This debugging code can be found in the code submitted. It is used to ensure the deviceList[] array was being properly updated for each of the functions altering it.

4 Reflection

Overall I would say the implementation was successful and the process was well planned and executed. However, there are some alternatives methods I could've used or features I could've implemented.

The maximum number of devices being ten, although a sensible amount, could have been increased, as plenty of the memory of the Arduino remained unused and therefore a larger number would have been possible. Furthermore, if I were to carry out this implementation individually, a dynamic array could be used which could consider the amount of free memory before adding a device to the deviceList[].

Although it works fully in the final implementation, the "data" (edited by "P-") aspect of the devices is slightly inefficient, as upon creating the methods I was unaware that there would be difficulties displaying the degrees and percentage sign. Whilst these issues were eventually dealt with, there is still some redundancy, with thermostat devices having "°C" at the end of their data despite only the first two characters being used when displaying. However, it is worth nothing this did improve readability when debugging, but still ultimately could've been improved on. I would fix this issue by removing the last two characters when the device type is equal to "T" and having only 3 characters when the device type is equal to "L" or "S".

Despite these faults, I still believe the implementation to be very successful and fulfil the requirements set by the task of acting as a dummy home monitor.

5 UDCHARS

Using the LCDs built in createChar() function, I defined 2 new characters to represent the upwards and downwards arrow. The downwards scroll arrow is stored in byte 0 and the upwards in bite 1.

The bitmap and design for downwards arrow:

```
0b00000,
0b00100,
0b00100,
0b00100,
0b00100,
0b10101,
0b01010,
0b00100
```

The bitmap and design for upwards arrow:

```
0b00100,
0b01010,
0b10101,
0b00100,
0b00100,
0b00100,
0b00100,
0b00000
```

The characters are created in the setup() function of the Arduino.

6 FREERAM

First, the line *extern int __heap_start, *__brkval;* declares two variables representing the start of the memory heap (heap_start) and the current end of the memory heap (brkval).

The function first checks if brkval is equal to 0, to check that any memory has been dynamically allocated. If it is equal, then the free space is from the start of the memory heap to the free variable.

If some memory has been dynamically allocated, the free space is set to the difference between the end of the memory heap and the address of the free variable. The size of the free variable is then added as this space should be free but is currently occupied by the variable.

The output of this function is displayed in selectPressed(), on the line beneath the student ID.