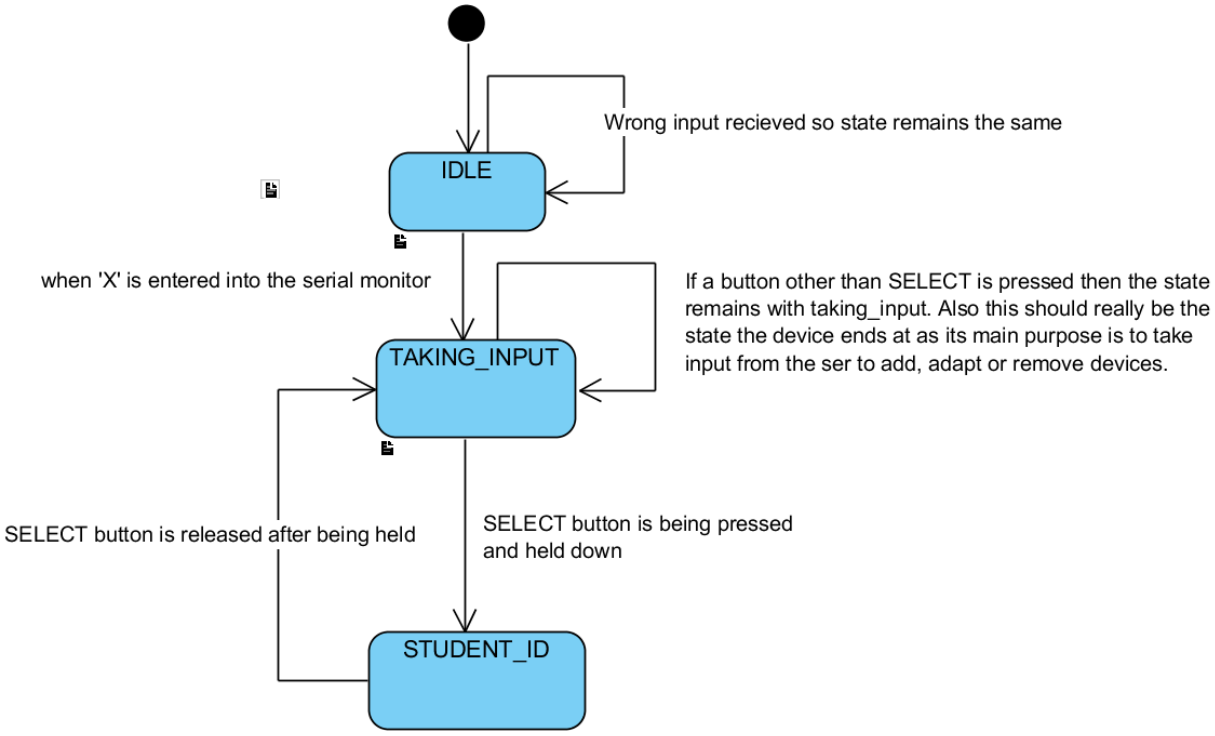
22COA202 Coursework

F225694

1 FSMs

I am using an FSM with 3 states => IDLE, TAKING_INPUT and STUDENT_ID:



I'm not sure how the fsm would end as it will always be active until power is turned off to the Arduino.

The three states:

- IDLE - When the machine is initially booted up it goes into its idle state where its waiting to be synchronized and initiated with a specific input.
- TAKING_INPUT - Once the value 'X' has been entered into the Serial monitor the TAKING_INPUT state is selected. The taking_input state is a very broad state as its all about taking input from the user, whether it's a button press or an input into the serial monitor, in this state its ready to receive all.
- STUDENT_ID – When the select button has been pressed and held down the states change to STUDENT_ID which outputs my student ID and the FREE SRAM available on the Arduino.

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.*

The following data structures are used in the code:

Structures:

- None

Enums:

- States - An enumeration that represents the various states of the finite state machine (FSM).

Classes:

- Wire - A class that provides an interface to the I2C bus.
- Adafruit_RGBLCDShield - A class that provides an interface to an RGB LCD shield.
- Adafruit_MCP23017 - A class that provides an interface to the MCP23017 I/O expander.

Constants:

- MAX_DEVICES - An integer constant that defines the maximum number of devices that can be stored in the devices array.
- the_devices - A string array that contains a list of valid device codes.
- downArrow - A byte array that defines the custom character, DOWN ARROW, for the LCD display.
- upArrow - A byte array that defines a custom character, UP ARROW, for the LCD display.

Variables:

- lcd - An instance of the Adafruit_RGBLCDShield class.
- MAX_DEVICES - An integer constant that defines the maximum number of devices that can be stored in the devices array.
- devices - A two-dimensional character array that stores the devices.
- num_devices - An integer variable that keeps track of the number of devices currently stored in the devices array.
- status1 - A two-dimensional character array that stores the status of each device, with the devices three-digit code to identify it.
- power1 - A two-dimensional character array that stores the power each device in the devices array, with the devices three-digit code to identify it.

- the_devices - A string array that contains a list of valid device codes.
- d_type - A character array that stores the device type code extracted from the input string.
- Present - A boolean variable that indicates whether the device type code extracted from the input string is valid.
- Sync - A boolean variable that is used to synchronize the LCD display and the serial output.
- currentIndex - An integer variable that is used to keep track of the current index in the devices array.
- Device - A character array that is used to store the device from the devices array.
- symbol1 - A character variable that is used to store the custom character, UP ARROW.
- symbol2 - A character variable that is used to store the custom character, DOWN ARROW.

The code defines several functions to update the global data structures/store:

- Device_type(char input1[]) - This function takes a character array input1 and extracts the device type from it, which is then stored in the d_type array. It also checks if the device type is present in the the_devices array and sets the present flag accordingly.

- adding(char input1[]) - This function adds a new device to the list of devices if it is valid. It calls Device_type to check if the device type is valid and checks for other conditions such as the format of the input string and the maximum number of devices allowed. If all conditions are met, it adds the device to the devices array, sets its status in the status1 array, and increments the num_devices variable. It also prints information about the added device and updates the LCD screen.

- freeRam() - This function calculates the amount of free memory on the microcontroller by subtracting the current stack pointer from either the heap start address or the current heap pointer, depending on whether the heap has been initialized. The result is returned as an integer.

3 Debugging

When using and assessing the code, there are somethings that I should let you know:

- the left button is used to print the first device to the screen and should be pressed after the status is changed in order to update the screen.
- Once the select button has been released it will print the first device from the devices array to the lcd screen.

4 Reflection

This was a very fun and interactive project and I think it went very well when considering my experience with programming. It started off tricky with many errors but when I properly understood the task ahead of me it made it much easier.

I am proud of my functions on the status and power as they took a while to understand what is the best way to go about doing this. It took many failed attempts but in the end I succeeded.

Something that I would definitely like to change is when the down button is pressed there is always an extra slide of symbols before 'invalid index' is printed onto the lcd screen. However, I've experimented with the code and never managed to fix it, so this will be present when you assess the code.

Another thing to consider is the naming conventions used in the code. Function names like 'status' and 'power' are not very descriptive and can make the code difficult to read and understand. This can be an issue for other developers as they will struggle to interpret the code correctly and accurately.

Overall, the code has some strengths in terms of functionality, but there are some areas for improvement, such as improving variable scope, naming conventions and polish up the outcome. These improvements can make the code more readable, maintainable, and easier to debug and modify.

Extension Features

5 UDCHARS

This extension was difficult to do and required research into how to print custom characters onto the lcd screen.

Each custom character is defined as a byte array with 8 elements, where each element represents a row of pixels in the character. Each row is represented as a binary number, where a pixel is turned on (or "filled") with a value of 1 and turned off (or "empty") with a value of 0.

To calculate the byte digits for the down arrow, I first visualized the shape of the arrow using a grid with 8 rows and 5 columns. Then, I filled in the pixels that should be turned on to create the arrow shape. Here's what the grid looks like for the up arrow:

0	0	0	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
1	1	1	1	1
0	1	1	1	0
0	0	1	0	0

In this grid, '1' represents a filled pixel, while '0' represents an empty pixel. I then converted each row to a binary number, starting from the top row and going down. For example, the first row is all empty, so its binary number is 0b00000. The second row has its 3rd column filled, so its binary number is 0b00100, and so on.

Once I had the binary numbers for each row, I combined them into a byte array by concatenating them together, starting from the top row and going down. For the down arrow, the resulting byte array is:

```
byte downArrow[8] = {
  0b00100,
  0b01110,
  0b11111,
  0b00100,
  0b00100,
  0b00100,
  0b00100,
  0b00000
};
```

This byte array represents the up arrow as an 8x5 grid of pixels, with each row represented as a binary number. Note that the byte digits are specified in binary notation by prefixing each row's binary number with 0b.

6 FREERAM

The freeRam() function computes the amount of free RAM available in the Arduino's memory. This is how it works:

- The function starts by declaring a variable free_memory to hold the amount of free RAM.
- It then checks whether the pointer __brkval is equal to zero. The __brkval pointer is used by the Arduino's memory allocation functions to keep track of the heap's end address.
- If __brkval is zero, then the heap is yet to be initialized, and free_memory is set to the difference between the address of free_memory and the start address of the heap, which is given by the __heap_start symbol.
- If __brkval is not zero, then the heap has already been initialized, and free_memory is set to the difference between the address of the free_memory variable and the current value of __brkval, which represents the end of the heap.
- The function then returns the value of free_memory.

The function calculates the amount of memory that is available for dynamic allocation by deducting the address of the free_memory variable from either the start or end address of the heap. The quantity that results is the amount of free RAM in bytes.

7 HCI

8 EEPROM

9 SCROLL

10 Submission

- *After following the instructions,* ***delete this section and ALL the subsequent sections from your report****.*
- Prepare the report as a PDF.

10.1 From Word source

If you have prepared this using the Word template, then use the styles `Heading 1` and `Heading 2` for each section and subsection. It should create a new page for each `Heading 1` and `Heading 2`. Please check this is the case.