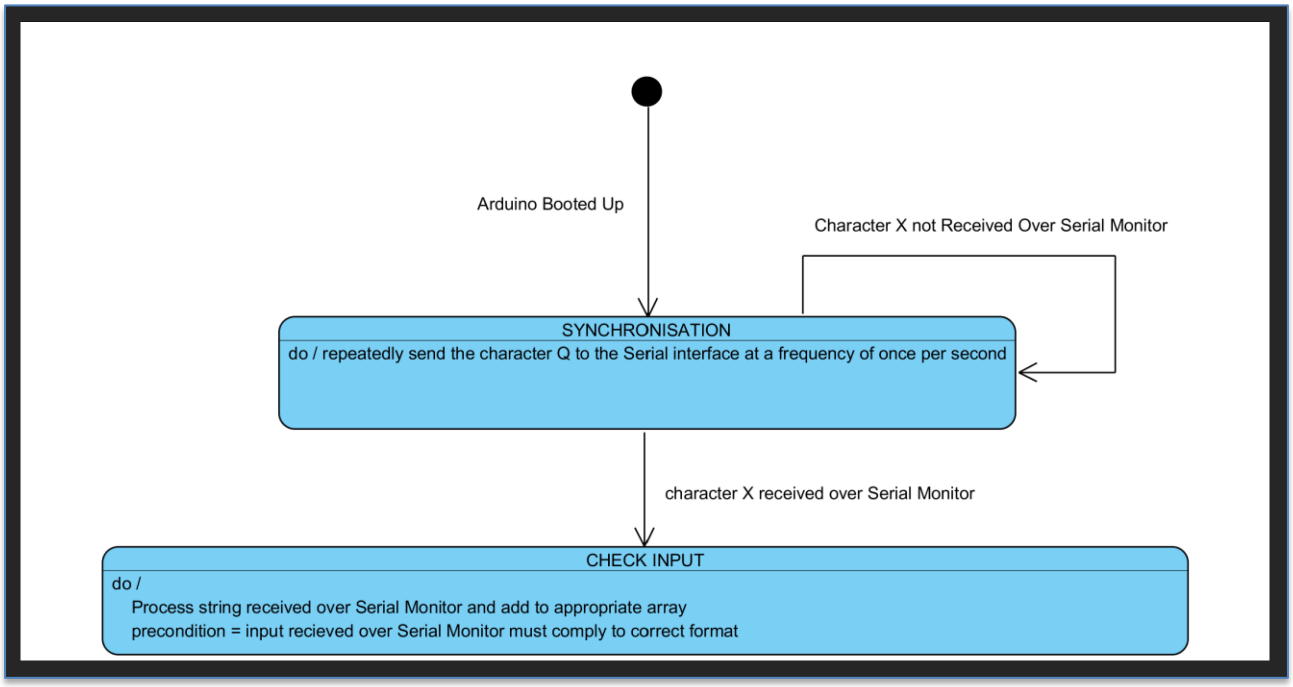
21COA202 Coursework

F116673

Semester 2 / SAP

1 FSMs

Finite State Machine 1


Arduino Booted Up
Character X not Received Over Serial Monitor
SYNCHRONISATION
do / repeatedly send the character Q to the Serial interface at a frequency of once per second
character X received over Serial Monitor
CHECK INPUT
do /
Process string received over Serial Monitor and add to appropriate array
precondition = input recieved over Serial Monitor must comply to correct format

In this finite state machine after the Arduino has booted up the Arduino enters a state of SYNCHRONISATION, in this state the Arduino reads any input over the serial monitor waiting for the character X to be received. Once this character is received the backlight is changed and the Arduino now enters a state of CHECK_INPUT.

This FSM is based off the following enumerated type:

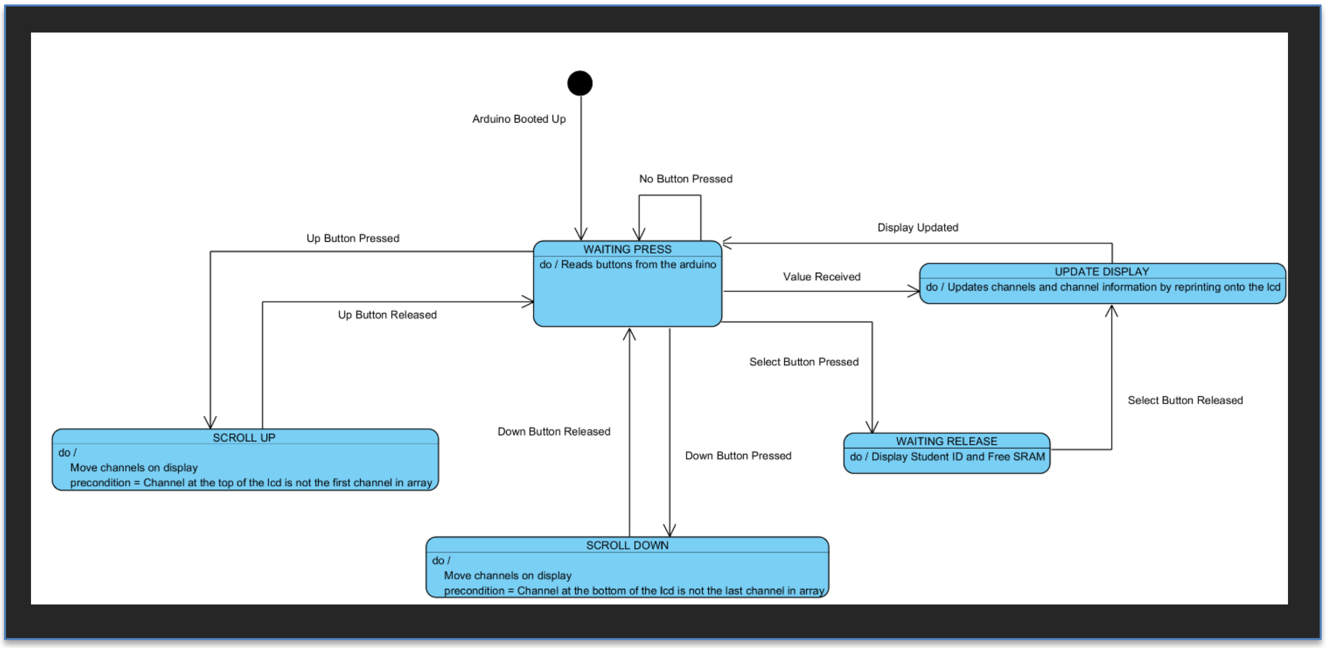
```
enum state_e {SYNCHRONISATION = 8, CHECK_INPUT};
```

In the state CHECK_INPUT (after there is an input available from the serial monitor):

- Inputs starting with the character "C" are either added to both channels and channels_for_display arrays, or the channels description is updated, or the channel is not added at all if the maximum number of channels allowed has been reached (refer to compromises made in RECENT extension).
- Arrays are sorted alphabetically.
- First 2 channels defined are printed onto the lcd and updated after every new value received.
- Inputs starting with either of the characters "X" or "N" are either used to update the maximum or minimum values for a channel in max_min_channels or is added to this array.
- Inputs starting with the character "V" are added to the values array up until the maximum number of values allowed has been reached.
- All Recent vales are checked to see if they are in range after new values are received by calling a function that checks if all recent values are in range. The function returns a byte that is used to update the backlight:

```
lcd.setBacklight(check_recent_values_range(channels_for_display, array_length, max_min_channels, max_min_channels_length, values, values_array_length));
```

Finite State Machine 2


Arduino Booted Up
No Button Pressed
Up Button Pressed
Display Updated
WAITING PRESS
do / Reads buttons from the arduino
Value Received
UPDATE DISPLAY
do / Updates channels and channel information by reprinting onto the lcd
Up Button Released
Select Button Pressed
Select Button Released
Down Button Released
Down Button Pressed
SCROLL UP
do /
Move channels on display
precondition = Channel at the top of the lcd is not the first channel in array
WAITING RELEASE
do / Display Student ID and Free SRAM
SCROLL DOWN
do /
Move channels on display
precondition = Channel at the bottom of the lcd is not the last channel in array

After the Arduino has booted up it enters a state of WAITING_PRESS where the Arduino continuously reads for button presses.

This FSM is based off the following enumerated type:

```
enum state_f {WAITING_PRESS = 8, UPDATE_DISPLAY, SCROLL_DOWN, SCROLL_UP, WAITING_RELEASE};
```

```
static byte channel_position_top = 0;
static byte channel_position_bottom = 1;
```

These two variables defined above are used to scroll the channels on the lcd.

These variables store the indexes (position) of the channels within the array channels_for_display for which we want to display at the top and bottom of the lcd. When we want to scroll through the channels on the lcd these two variables are either incremented or decremented by the same amount depending on which way the user wants to scroll. Printing the channels at these two indexes after incrementing or decrementing will then shift the channels on the lcd.

If the top button is pressed, the Arduino enters the state SCROLL_UP where:

- If the variable channel_position_top is 0, meaning the index (position) of the channel in the array is at the top, then there no more channels to scroll up through. The state changes back to WAITING_PRESS when the user releases the button.
- If the variable channel_position_top is greater than 0, then this variable and channel_position_bottom are both decremented by 1. After this the channels at these position in the array channels_for_display are printed onto the lcd. The appropriate arrows are printed, as well as calling the following functions which will display the averages, names and scroll the names if needed for the channels now printed onto the lcd:

```
display_channel_names_and_avg(channels, values, values_array_length, array_length, channels_for_display, channel_position_bottom, channel_position_top);
scroll_top_channel = scroll_top_channel_bool(channels, array_length, channels_for_display, channel_position_top);
scroll_bottom_channel = scroll_bottom_channel_bool(channels, array_length, channels_for_display, channel_position_bottom);
```

After the up button is released, the Arduino returns to the state WAITING_PRESS.

The same occurs when the down button is pressed, but this time if the variable channel_position_bottom is less than the number of channels in the array channels_for_display (there are more channels to scroll down through in the array) the variables channel_position_bottom and channel_position_up are both incremented by 1.

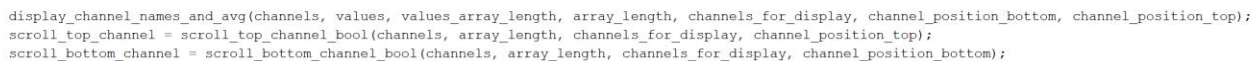
If the select button is pressed, the Arduino enters the state WAITING_RELEASE and stores the time at which the select button was pressed into the variable press_time by using the millis function: `press_time = millis();`

In WAITING_RELEASE:

- Using the millis function the current time is compared to the press_time variable which will determine the time passed since the select button was pressed.
- If the time passed since pressing the select button is greater than or equal to one second, my student number is displayed as well as the amount of free SRAM.
- When the select button is released the lcd is cleared and the state changes to UPDATE_DISPLAY, after which the state returns to WAITING_PRESS.

When the Arduino receives a new value over the serial monitor the state UPDATE_DISPLAY is entered where:

- The lcd is cleared and the channels are reprinted from the array channels_for_display at the indexes defined by channel_position_bottom and channel_position_bottom. The appropriate arrows are printed along with calling the following functions which will display the averages, names and scroll the names if needed for the channels now printed onto the lcd:

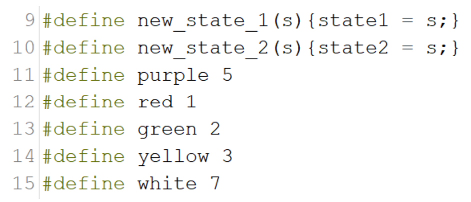


```
display_channel_names_and_avg(channels, values, values_array_length, array_length, channels_for_display, channel_position_bottom, channel_position_top);
scroll_top_channel = scroll_top_channel_bool(channels, array_length, channels_for_display, channel_position_top);
scroll_bottom_channel = scroll_bottom_channel_bool(channels, array_length, channels_for_display, channel_position_bottom);
```

2 Data structures

Data Structures:

#define

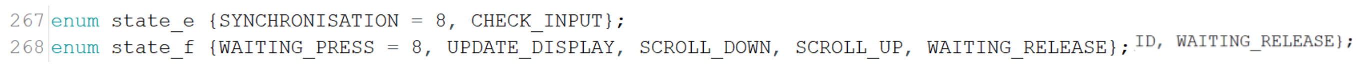


```
#define new_state_1(s){state1 = s;}
#define new_state_2(s){state2 = s;}
#define purple 5
#define red 1
#define green 2
#define yellow 3
#define white 7
```

Using the #define statement, I have been able to define functions and constants for increased readability within my code.

- I have defined 2 functions which update the static enumerated type variables state1 and state2.
- I have defined colour names with the corresponding byte that indicates that colour. I can now use the name of the colour when I want to provide the byte for that colour within my code, which gives more meaning to my code and increases the readability.

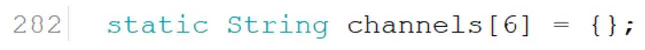
enumerated types



```
enum state_e {SYNCHRONISATION = 8, CHECK_INPUT};
enum state_f {WAITING_PRESS = 8, UPDATE_DISPLAY, SCROLL_DOWN, SCROLL_UP, WAITING_RELEASE}; ID, WAITING_RELEASE};
```

Using enumerated types, two sets have been created consisting of named variables (enumerators). Each enumerator is assigned a value, by assigning the first enumerator a value of 8.

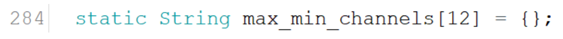
Arrays



```
static String channels[6] = {};
```

This static array of type string is used to store the channels defined and received over the Serial Monitor, this includes the channel letter and description as a string.

An example of the contents of this array: ["CAmain"," CBSecondary"]



```
static String max_min_channels[12] = {};
```

This static array of type string is used to store any maximum or minimum value (any input that starts with 'X' or 'N', has a declared channel and is in the correct format) for a channel received over the serial monitor. This array is then iterated through in the program when any new value is received over the serial monitor to check if it exceeds any limits set for its channel. New maximum or minimum values received for a channel are replaced in this array.

```
static String channels_for_display [6] = {};
```

This static array or type string is used to store all the defined channels like the array channels. However, every time a value is accepted over the serial monitor the corresponding channel in this array is edited so that to have the recent value next to the channel letter, as a result this array stores the channel letters and recent values (each channel letter and recent value is a single string). Pressing the up and down buttons on the Arduino will move up and down through this array printing two strings at a time onto the lcd.

```
static String values [36] = {};
```

This static array of type string is used to store values received and accepted over the serial monitor. This array is iterated through in the program to determine the averages for specific channels. This is done by comparing the channel letter of the desired channel we want the average for to the channel letters of the values in this array.

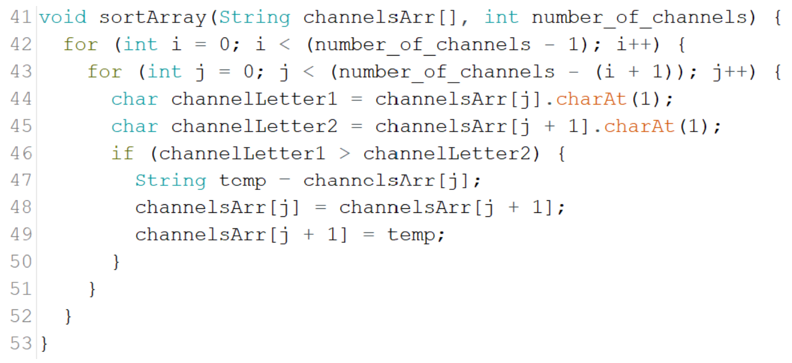
Functions updating global structures

- void add_most_recent_value(String list[], int number_of_channels, String value)

```
void add_most_recent_value(String list[], int number_of_channels,  String value) {
  for (int i = 0; i < number_of_channels; i++) {
    if (list[i].substring(1, 2) == value.substring(1, 2)) {
      switch (value.length()) {
        case 5:
          list[i] = list[i].substring(0, 2) + value.substring(2, 5);
          break;
        case 4:
          list[i] = list[i].substring(0, 2) + value.substring(2, 4);
          break;
        case 3:
          list[i] = list[i].substring(0, 2) + value.substring(2, 3);
          break;
      }
    }
  }
}
```

This function takes in 3 parameters, iterates through an array of strings which in this case will be the array used for displaying the channels and recent values on the lcd, and compares the channel letter of the most recent value received over the Serial Monitor (String value) to the channel letters in the array. After finding a matching channel in the array, this channel string is edited so that it contains this most recent value.

- void sortArray(String channelsArr[], int number_of_channels)



```
void sortArray(String channelsArr[], int number_of_channels) {
  for (int i = 0; i < (number_of_channels - 1); i++) {
    for (int j = 0; j < (number_of_channels - (i + 1)); j++) {
      char channelLetter1 = channelsArr[j].charAt(1);
      char channelLetter2 = channelsArr[j + 1].charAt(1);
      if (channelLetter1 > channelLetter2) {
        String temp = channelsArr[j];
        channelsArr[j] = channelsArr[j + 1];
        channelsArr[j + 1] = temp;
      }
    }
  }
}
```

Using bubble sort this function is used to sort the channels into alphabetical order by the channel letters.

3 Debugging

In the following code below, I have defined a serial print statement that I have used throughout my code to print information to the serial monitor for debugging. Defining the serial print statement has increased the readability of my code.

```
#define DEBUG
#ifdef DEBUG
#define debug_print(x) Serial.println(x)
#else
#define debug_print(x)
#endif
```

The following code below uses for loops to iterate through all the arrays in my program displaying all the strings in these arrays onto the serial monitor to aid in debugging.

```
        debug_print(F("---------------------------------"));
        debug_print(F("-----------channels--------------"));
        debug_print(F("---------------------------------"));
        for (int i = 0; i <= array_length; i++) {
          Serial.println(channels[i]);
        }
        debug_print(F("---------------------------------"));
        debug_print(F("------channels for display-------"));
        debug_print(F("---------------------------------"));
        for (int i = 0; i < number_of_channels_for_display; i++)
          Serial.println(channels_for_display[i]);
        }
        debug_print(F("---------------------------------"));
        debug_print(F("-------------max/min-------------"));
        debug_print(F("---------------------------------"));
        for (int i = 0; i < max_min_channels_length; i++) {
          Serial.println(max_min_channels[i]);
        }
        debug_print(F("---------------------------------"));
        debug_print(F("-------------values--------------"));
        debug_print(F("---------------------------------"));
        for (int i = 0; i < values_array_length; i++) {
          Serial.println(values[i]);
        }
```

4 Reflection

Through the completion of this coursework, I came to realize the importance of memory optimization. Towards the completion of my program, I noticed that the heavy use of arrays used up a lot of memory. Resulting from this I was unable to implement the HCI extension feature as I did not have enough memory. I also did not complete the EEPROM extension as I struggled to understand how to implement this extension.

However, if I had enough memory to implement HCI I would have created a data structure, preferably an array. This array would have stored the channels with a recent value over their set maximums or channels with recent values below their set minimums depending on which button was pressed. I would have then defined two functions, both would iterate through the array which stores all the channels, one function would add channels to the array previously mentioned where the channels recent value is above the maximum, and the other function adding channels with recent values below the minimum to the same array.

In continuation, when the left button is pressed and released a new state would have been entered where one of the functions mentioned above would have been called setting up an array to be scrolled through. Similar code would have been implemented into this state to that used to scroll up and down through the channels for basic. The same would occur when pressing and releasing the right button, but with a different function call. After either the left or right button is pressed and released again the array which had stored the channels for this extension would be cleared for next time, and Arduino would go into the UPDATE_DISPLAY state which would return the display to normal.

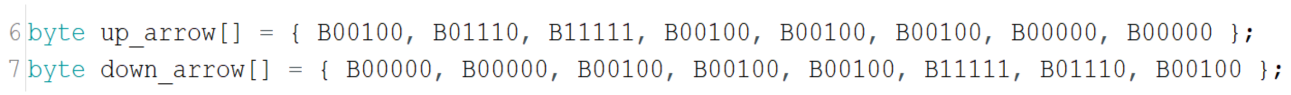
Due to implementing the RECENT extension I had to make a compromise on the number of channels stored to prevent my program from running out of memory, I thought it was more important for my program to remain functional rather than to store the correct number of channels and risk my program crashing.

On a good note, there is some functionality I am particularly proud of. The first is that my function that checks if all recent values are in range considers if the user sets the maximum for a channel below the minimum. In this situation the maximum is always prioritized in terms of changing the backlight. For example, if a recent value is below the maximum and minimum for a channel nothing happens. However, if the recent value is above the maximum and below the minimum then the backlight is changed to red. Some extra flexibility added to my program is that the maximum and minimum values for a channel can be updated.

Finally looking back over my coursework if there had been something I had done differently it would have been to use structures instead of arrays to store data, as structures would have used less memory.

5 UDCHARS

Location of defined characters



```
6 byte up_arrow[] = { B00100, B01110, B11111, B00100, B00100, B00100, B00000, B00000 };
7 byte down_arrow[] = { B00000, B00000, B00100, B00100, B00100, B11111, B01110, B00100 };
```

Lines 6 and 7 define my characters.

I used the following website in helping define these special characters:

https://chareditor.com/

Changes to FSM

Changes included adding extra code to the following states:

- SCROLL_UP, SCROLL_DOWN: arrows are printed every time the channels are moved up and down, depending on the indexes of the channels on display in the array channels_for_display, the down and up arrows are printed accordingly.
- CHECK_INPUT: down arrow printed once third channel defined in this state.
- DISPLAY_VALUE: depending on the indexes of the channels in the array channels_for_display which are to be displayed, the down and up arrows are printed accordingly.

When implementing UDCHARS into my code, I first created the characters using the lcd.createChar() function and then used the lcd.write() function to display the characters onto the lcd. Using conditional statements to see whether the channel information being displayed on the screen is the first or last string in the array channels_for_display, I displayed the appropriate characters accordingly.

For example:

```
if (channel_position_top == 0) {
  lcd.clear();
  lcd.setCursor(0, 0);
  String str1 = value_format(channels_for_display[channel_position_top]);
  lcd.print(str1);

  lcd.createChar(0, down_arrow);
  lcd.setCursor(0, 1);
  lcd.write(0);
  lcd.setCursor(1, 1);
  String str2 = value_format(channels_for_display[channel_position_bottom]);
  lcd.print(str2);
```

Here this if statement checks whether the channel information in the top row of the lcd is the first element within the array, and if so then there are no more channels to scroll upwards, and as a result the up_arrow character is not displayed however the down_arrow character is, as to indicate there are more channels to scroll through below.

```
lcd.createChar(0, up_arrow);
lcd.write(0);
```

To display the character defined by up_arrow in the position (0,0) on the lcd screen, the lcd.setCursor() function does not have to be used as position (0,0) is default.

```
lcd.createChar(1, down_arrow);
lcd.setCursor(0, 1);
lcd.write(1);
```

However, for displaying the down_arrow character which will not be displayed in position (0,0), the lcd. setCursor() function needs to be used before writing the character to the lcd to specify the character's location on the lcd.

6 FREERAM

Changes to FSM

The changes to the FSM were adding this section of code below to the WAITING_RELEASE state so that it can be executed along with the code to display my student number once the select button has been held down for longer than one second.

```
        lcd.setCursor(0, 1);
        lcd.print("SRAM: ");
        lcd.setCursor(6, 1);
        lcd.print(freeMemory());//display free SRAM
```

This code above prints the text ‘SRAM: ’ on the bottom row of the lcd and then next to this text prints the result of the function call freeMemory() which returns the amount of free SRAM available at that point in time.

Code needed to display free SRAM

```
#ifdef __arm__
// should use uinstd.h to define sbrk but Due causes a conflict
extern "C" char* sbrk(int incr);
#else // __ARM__
extern char *__brkval;
#endif // __arm__

int freeMemory() {
  char top;
#ifdef __arm__
  return &top - reinterpret_cast<char*>(sbrk(0));
#elif defined(CORE_TEENSY) || (ARDUINO > 103 && ARDUINO != 151)
  return &top - __brkval;
#else // __arm__
  return __brkval ? &top - __brkval : &top - __malloc_heap_start;
#endif // __arm__
}
```

On the left is the code and function used to return the amount of free SRAM, which is located before the main void loop ().

Code reference:

Phillips, I (2022) 21CO202: Embedded Systems Programming Understanding Memory [Source code]. https://learn.lboro.ac.uk/pluginfile.php/1815191/mod_resource/content/4/esp-ws3.pdf

9 RECENT

Compromises made

I have had to reduce the number of channels that can be defined to 6 channels rather than 26 (26 letters in alphabet). The reason for this is if I store more channels while implementing the RECENT extension where I need to store values the program crashes and does not work correctly as too much memory is used.

Subsequently the array which stores the maximum and minimum values for the channels can only store 12 of these values, a minimum and maximum for all 6 channels.

Resulting from this compromise I have been able to define an array of size 36 which will store and calculate an average for the first 6 values for each channel. Any array size larger than this will use up too much memory. After each channel has received its six values the average will remain the same on the lcd but the recent values next to the channel letters will still change.

For all my code and functionality to run and work correctly due to the usage of storage I have had to choose to limit the number of channels, as well as the number of values used to calculate the averages for these channels. Otherwise, my program will run out of memory and crash. This is a compromise I have decided to make to prevent my program from running out of memory.

I rather make sure that with all possible inputs and number of inputs my functionality will remain working, I could have stored more channels and values per channel but after inputting a certain number of values my Arduino runs out of memory.

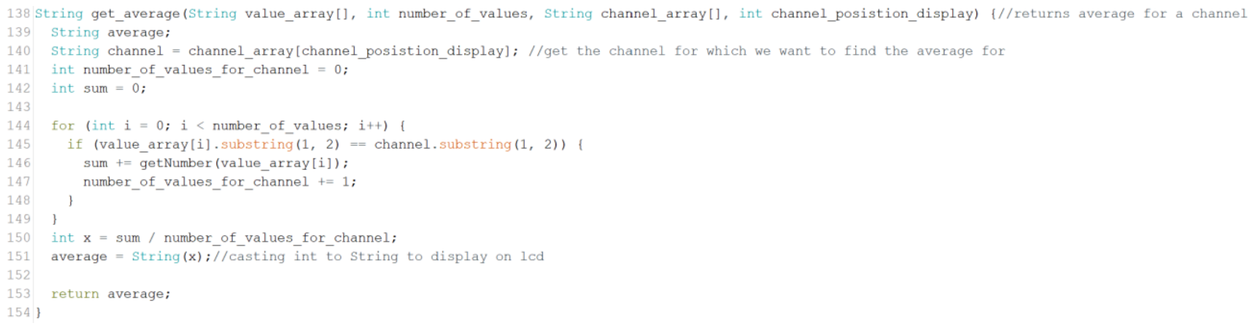
Code needed to calculate averages

```
static String values [36] = {};
static byte values_array_length = 0;
```

An array is used to store the first 6 recent values for all 6 channels. The name of the array is values. The array is declared static so that not to lose the values stored in the array while looping through the main loop.

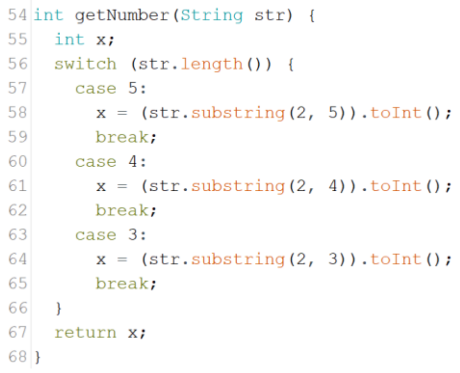
This Data structure is declared after the main void loop () of the program.

Additionally, a static variable of type integer named values_array_length is also declared within the same main loop which indicates the number of values currently within the array values. This variable is incremented every time a new value is accepted by the program across the serial monitor.



```
String get_average(String value_array[], int number_of_values, String channel_array[], int channel_posistion_display) {//returns average for a channel
  String average;
  String channel = channel_array[channel_posistion_display]; //get the channel for which we want to find the average for
  int number_of_values_for_channel = 0;
  int sum = 0;

  for (int i = 0; i < number_of_values; i++) {
    if (value_array[i].substring(1, 2) == channel.substring(1, 2)) {
      sum += getNumber(value_array[i]);
      number_of_values_for_channel += 1;
    }
  }
  int x = sum / number_of_values_for_channel;
  average = String(x);//casting int to String to display on lcd

  return average;
}
```

This function get average above will determine and return the average of all the values for a specific channel in this array values. The parameter channel_position_display is of type integer and is used to determine the channel we want to find the average for by providing the position of the channel in the 3rd parameter channel_array []. This function iterates through the values array finding values that have the same channel letter as the channel we are trying to find the average for. If we find a value that has the channel letter we are looking for, using the function getNumber(String str) which returns an integer value, the function adds this to the variable sum and increments the variable number_of_values_for_channel by one. After the whole array has been iterated through the average is calculated by dividing the sum of values for that channel by the number of values that channel has.



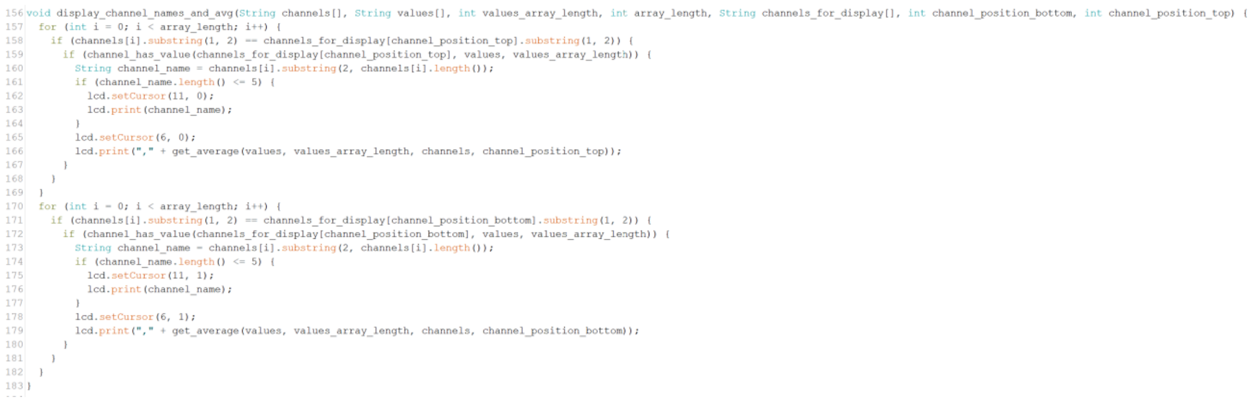
```
int getNumber(String str) {
  int x;
  switch (str.length()) {
    case 5:
      x = (str.substring(2, 5)).toInt();
      break;
    case 4:
      x = (str.substring(2, 4)).toInt();
      break;
    case 3:
      x = (str.substring(2, 3)).toInt();
      break;
  }
  return x;
}
```

This function to the left which is used within the get_average() function takes in a single parameter of type string and returns an integer. By using .substring(), the function casts part of a channels value of type string (e.g. VA45) to an integer depending on the strings length as some values include 1,2 or 3 digits.

The following function dislay_channel_names_and_avg() below is used not only to display the names of the channels on the lcd but to also display the averages. This is done by setting the lcd cursor to position (6,0) for the channel at the top of the lcd, and then by printing the result of concatenating a comma to the returned string (so that the information displayed on the lcd is in the correct format) from the function call:

get_average(values, values_array_length, channels, channel_position_top)

The same is done for the channel at the bottom of the lcd by setting the cursor to position (6,1), and by replacing the parameter channel_position_top to channel_position_bottom in the function call which indicates the index of the channel we want in the array channels_for_dsiplay.



```
void display_channel_names_and_avg(String channels[], String values[], int values_array_length, int array_length, String channels_for_display[], int channel_position_bottom, int channel_position_top) {
  for (int i = 0; i < array_length; i++) {
    if (channels[i].substring(1, 2) == channels_for_display[channel_position_top].substring(1, 2)) {
      if (channel_has_value(channels_for_display[channel_position_top], values, values_array_length)) {
        String channel_name = channels[i].substring(2, channels[i].length());
        if (channel_name.length() <= 5) {
          lcd.setCursor(11, 0);
          lcd.print(channel_name);
        }
        lcd.setCursor(6, 0);
        lcd.print("," + get_average(values, values_array_length, channels, channel_position_top));
      }
    }
  }
  for (int i = 0; i < array_length; i++) {
    if (channels[i].substring(1, 2) == channels_for_display[channel_position_bottom].substring(1, 2)) {
      if (channel_has_value(channels_for_display[channel_position_bottom], values, values_array_length)) {
        String channel_name = channels[i].substring(2, channels[i].length());
        if (channel_name.length() <= 5) {
          lcd.setCursor(11, 1);
          lcd.print(channel_name);
        }
        lcd.setCursor(6, 1);
        lcd.print("," + get_average(values, values_array_length, channels, channel_position_bottom));
      }
    }
  }
}
```

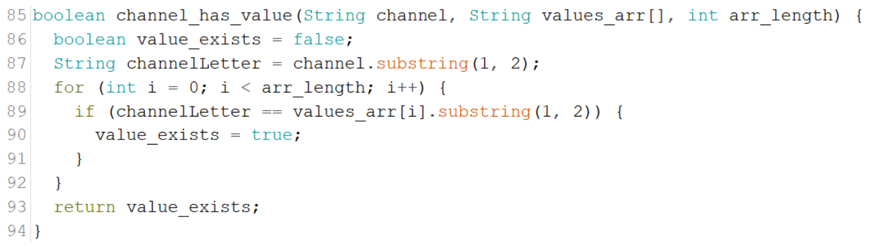
The code that which calls the function returning the average for a channel is only executed if the function call:

channel_has_value(channels_for_display[channel_position_top], values, values_array_length)

OR

channel_has_value(channels_for_display[channel_position_bottom], values, values_array_length)

returns true.

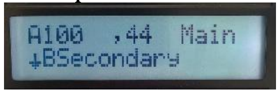


```
boolean channel_has_value(String channel, String values_arr[], int arr_length) {
  boolean value_exists = false;
  String channelLetter = channel.substring(1, 2);
  for (int i = 0; i < arr_length; i++) {
    if (channelLetter == values_arr[i].substring(1, 2)) {
      value_exists = true;
    }
  }
  return value_exists;
}
```

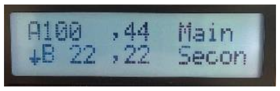
This function directly above checks to see if a channel has yet received a value. This is done by iterating through the values array comparing each value's channel letter to the channel letter of the channel we want to find if it's received a value yet. If a value is found to have the same channel letter as that of the channel we are doing the search for, the function returns true.

The purpose of this function is that we want to know when a channel has received its first value so that code can be executed to display the channels information (channel average). There is no point trying to calculate a channels average until it receives a value, that's why this function is important in using it as a Boolean to check whether a certain channels information must be displayed or not.

Example:



Before the channel BSecondary receives a value



After the channel BSecondary receives a value

The following function below checks to see whether the maximum number of values allowed to be stored for a channel has been reached. The purpose of this is so that the values array does not try store more values than what its size is defined to store. Additionally, it controls and makes sure that each channel will only have a maximum of 6 values stored for the average calculations.

```
boolean add_value_array(String values[], int values_array_length, String value) {
  int count = 0;//creates a count of the number of values currently defined for a channel
  boolean add_channel = false;
  for (int i = 0; i < values_array_length; i++) {
    if (values[i].substring(1, 2) == value.substring(1, 2)) { //check to see if both have the same channel letter
      count += 1;
    }
  }
  if (count < 4) {//check to see if the maximum number of values allowed to be stored for channel has been reached
    add_channel = true;
  } else {
    debug_print("Maximum values for that channel reached for RECENT extension");
  }

  return add_channel;
}
```

The function compares the channel letter of a value which is a parameter, to the channel letters of the values in the array values, a count is incremented every time a value is found to have the same channel letter in the array, if the count is less than 4, then that channel has not received the maximum 6 values that can be stored for this channel. The function then returns a Boolean value true which is used to determine is a value can be added to the values array.

Changes to FSM

The only change to the FSM is adding this line of code below to the SCROLL_UP, SCROLL_DOWN and DISPLAY_VALUE states after the code that prints the two strings from the channels_for_display array onto the top and bottom of the lcd.

```
        display_channel_names_and_avg(channels, values, values_array_length, array_length, channels_for_display, channel_position_bottom, channel_position_top);
```

Another change to the FSM was adding the following code to the CHECK_INPUT state:

```
if (add_value_array(values, values_array_length, input)) { //check if value can be added for determining average
  values[values_array_length] = input;
  values_array_length += 1;
}
```

This code calls a function which returns a Boolean, if the function call returns true then the value received over the serial monitor is added to the array values.

10 NAMES

Code needed to display channel names

```
static String channels[6] = {};
static byte array_length = 0;
```

An Array channels which stores values of type String of set length 6 of type static is used to store the declared channels and their names. The variable array_length of type integer declared static is the current length of the array channels (it is incremented every time a new channel is defined) and is used to loop through the array channels so that to find the correct channel name to display for a given channel letter.

```
void display_channel_names_and_avg(String channels[], String values[], int values_array_length, int array_length, String channels_for_display[], int channel_position_bottom, int channel_position_top) {
  for (int i = 0; i < array_length; i++) {
    if (channels[i].substring(1, 2) == channels_for_display[channel_position_top].substring(1, 2)) {
      if (channels[i].substring(2, channels[i].length()) != channels_for_display[channel_position_top].substring(2, channels_for_display[channel_position_top].length())) {
        lcd.setCursor(11, 0);
        String channel_name = channels[i].substring(2, channels[i].length());
        lcd.print(channel_name);

        lcd.setCursor(6, 0);
        lcd.print("," + get_average(values, values_array_length, channels, channel_position_top));
      }
    }
  }
  for (int i = 0; i < array_length; i++) {
    if (channels[i].substring(1, 2) == channels_for_display[channel_position_bottom].substring(1, 2)) {
      if (channels[i].substring(2, channels[i].length()) != channels_for_display[channel_position_bottom].substring(2, channels_for_display[channel_position_bottom].length())) {
        lcd.setCursor(11, 1);
        String channel_name = channels[i].substring(2, channels[i].length());
        lcd.print(channel_name);

        lcd.setCursor(6, 1);
        lcd.print("," + get_average(values, values_array_length, channels, channel_position_bottom));
      }
    }
  }
}
```

This function above is used to display the channel names and averages for the two channels currently on the lcd.

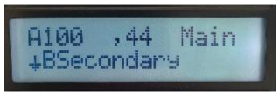
Firstly, using two for loops (one for each channel on display) the function iterates through the array channels and compares each channel letter in the array to the channel letters of the channels at indexes channel_position_top and channel_position_bottom in array channels_for_display. If the channel letters are found to be equal, then we know we have found the correct channel and channel name within the array channels, and we can use this to print the channel name for the channel and recent value being displayed at the top and bottom of the lcd.

```
boolean channel_has_value(String channel, String values_arr[], int arr_length) {
  boolean value_exists = false;
  String channelLetter = channel.substring(1, 2);
  for (int i = 0; i < arr_length; i++) {
    if (channelLetter == values_arr[i].substring(1, 2)) {
      value_exists = true;
    }
  }
  return value_exists;
}
```

This function directly above checks to see if a channel has yet received a value. The purpose of this function is that we want to know when a channel has received a value for the first time so that code can be executed to display the channels information (channel average and name).

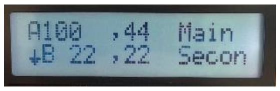
This function is used in an if statement to execute the code that displays the channels name only if the channel has received a value (no point in displaying the channel name a second time until the channel receives a value).

If the channel does have a value, then grab the channel description using the substring function from position 2 till the end of the string (grabbing everything except for "C" and the channel letter) which then is printed onto the lcd at the appropriate position. For example:



```
A100  ,44  Main
↓BSecondary
```

Before the channel BSecondary receives a value



```
A100  ,44  Main
↓B 22 ,22  Secon
```

After the channel BSecondary receives a value

Changes to FSM

This function below is called within the states of SCROLL_UP, SCROLL_DOWN and DISPLAY_VALUE in the FSM, so that every time a button is pressed, and the channels move position on the lcd, the average and name is updated for the appropriate two channels currently on display.

Additionally, this function is called within the CHECK_INPUT state so that incase the channels are not scrolled the averages are updated after receiving values. Therefor the averages can be updated live rather than having to wait for a button press to update the averages of the channels.

```
        display_channel_names_and_avg(channels, values, values_array_length, array_length, channels_for_display, channel_position_bottom, channel_position_top);
```

11 SCROLL

Required Parts and changes to FSM

The main code that scrolls the names of channels that are too long needs to be looped through continuously, as a result the main code to scroll the channel names for the top and bottom channels on the lcd is situated at the end of the state WAITING_PRESS. This is because this is a state that the Arduino stays in most of the time, so it is looped through continuously compared to the other states of the FSM where the code for those states is looped through once and then the state is returned to WAITING_PRESS.

The code below is what is placed at the end of the WAITING_PRESS state.

The two main if statements that check to see if the Booleans scroll_top_channel and scroll_bottom_channel are true contain code that scrolls the names for the channels on display by scrolling two characters of the name to the left every 2 seconds. This is done by using another if statement checking to see if 500 milli seconds have passed by comparing the time from starting the iteration of the loop (now) to the current time using millis().

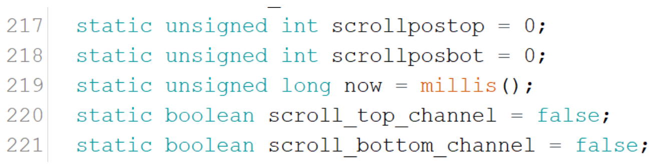
```
    if (scroll_top_channel == true) {
      String scroller_channel = get_channel_name(channels, array_length, channels_for_display, channel_position_top) + " ";

      if (millis() - now > 500) {
        now = millis();
        scrollpostop ++;
        if (scrollpostop >= scroller_channel.length()) {
          scrollpostop = 0;
        }
      }
      lcd.setCursor(11, 0);
      lcd.print(scroller_channel.substring(scrollpostop, scrollpostop + 16));
      lcd.setCursor(11, 0);
    }

    if (scroll_bottom_channel == true) {
      String scroller_channel = get_channel_name(channels, array_length, channels_for_display, channel_position_bottom) + " ";

      if (millis() - now > 500) {
        now = millis();
        scrollposbot ++;
        if (scrollposbot >= scroller_channel.length()) {
          scrollposbot = 0;
        }
      }

      lcd.setCursor(11, 1);
      lcd.print(scroller_channel.substring(scrollposbot, scrollposbot + 16));
      lcd.setCursor(11, 1);
    }
```

The code below is also added to the FSM, and is located in the SCROLL_UP, SCROLL_DOWN and UPDATE_DISPLAY states after the function call that displays the channel names and averages is executed(display_channel_names_and_avg()). These two lines of code set the two Booleans scroll_top_channel and scroll_bottom_channel by calling two functions that return Booleans.

```
      scroll_top_channel = scroll_top_channel_bool(channels, array_length, channels_for_display, channel_position_top);
      scroll_bottom_channel = scroll_bottom_channel_bool(channels, array_length, channels_for_display, channel_position_bottom);
```

Functions and lines of code needed

To the right is some code defined after void loop():



```
static unsigned int scrollpostop = 0;
static unsigned int scrollposbot = 0;
static unsigned long now = millis();
static boolean scroll_top_channel = false;
static boolean scroll_bottom_channel = false;
```

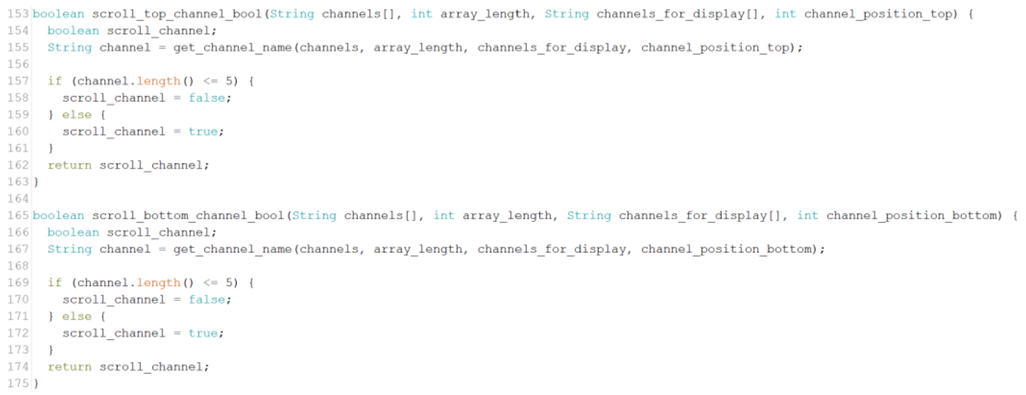
- The two variables scrollpostop and scrollposbot of type unsigned int are declared static and are needed to be able to scroll through the channel names. Incase both channels on display have names that need to be scrolled (of which both names could have different lengths) it is needed to have separate scrolling code for both the top and bottom channels on the lcd (code provided above) and these two variables which store the position (an integer indicating a letter in the string) from where the string is being printed, so that to know when the whole string has been scrolled through.

- The variable now of type unsigned long which is declared static stores the current time since the start of the program and is updated every time the program iterates through the main void loop().

- The two Booleans scroll_top_channel and scroll_bottom_channel is used to tell whether a channel name is too long and needs to be scrolled on the lcd.

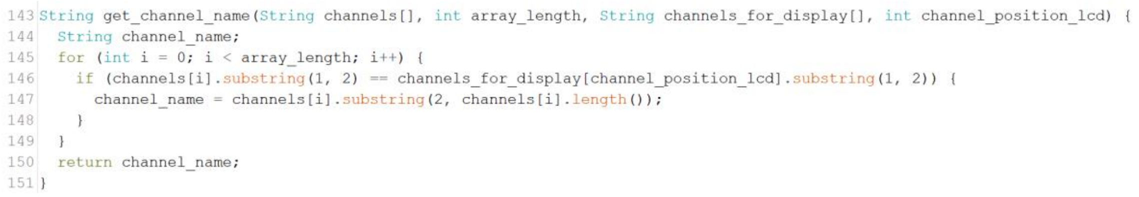
Functions needed

Below are two functions that return a Boolean value and are called to set two Booleans needed to determine whether the channel names on display need to be scrolled. Due to the space that the AVG extension takes up on the lcd, there are only 5 spaces left to display the channels names, so any channel name with length longer than 5 needs to be scrolled.



```
boolean scroll_top_channel_bool(String channels[], int array_length, String channels_for_display[], int channel_position_top) {
  boolean scroll_channel;
  String channel = get_channel_name(channels, array_length, channels_for_display, channel_position_top);

  if (channel.length() <= 5) {
    scroll_channel = false;
  } else {
    scroll_channel = true;
  }
  return scroll_channel;
}

boolean scroll_bottom_channel_bool(String channels[], int array_length, String channels_for_display[], int channel_position_bottom) {
  boolean scroll_channel;
  String channel = get_channel_name(channels, array_length, channels_for_display, channel_position_bottom);

  if (channel.length() <= 5) {
    scroll_channel = false;
  } else {
    scroll_channel = true;
  }
  return scroll_channel;
}
```

The function get_channel_name() below returns a string which is the channel name for either the channel at the top or bottom of the lcd so that the length can be checked.



```
String get_channel_name(String channels[], int array_length, String channels_for_display[], int channel_position_lcd) {
  String channel_name;
  for (int i = 0; i < array_length; i++) {
    if (channels[i].substring(1, 2) == channels_for_display[channel_position_lcd].substring(1, 2)) {
      channel_name = channels[i].substring(2, channels[i].length());
    }
  }
  return channel_name;
}
```