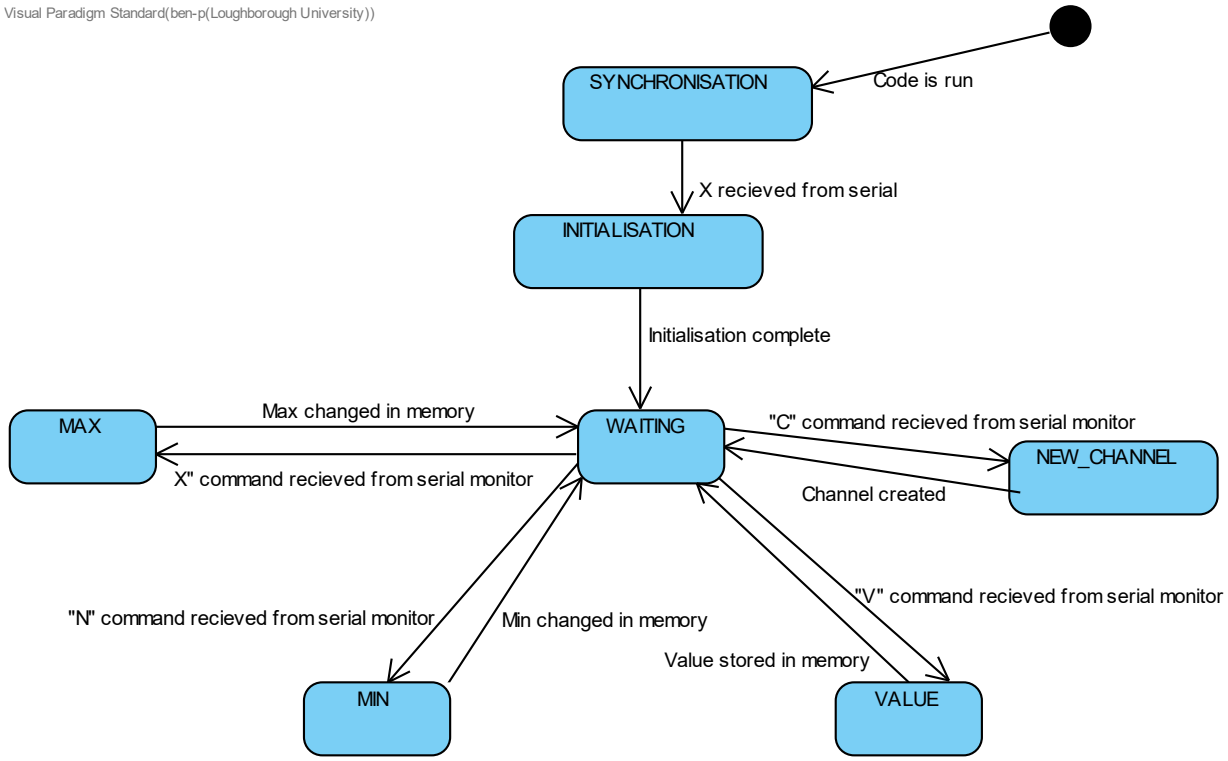
21COA202 Coursework

F128493

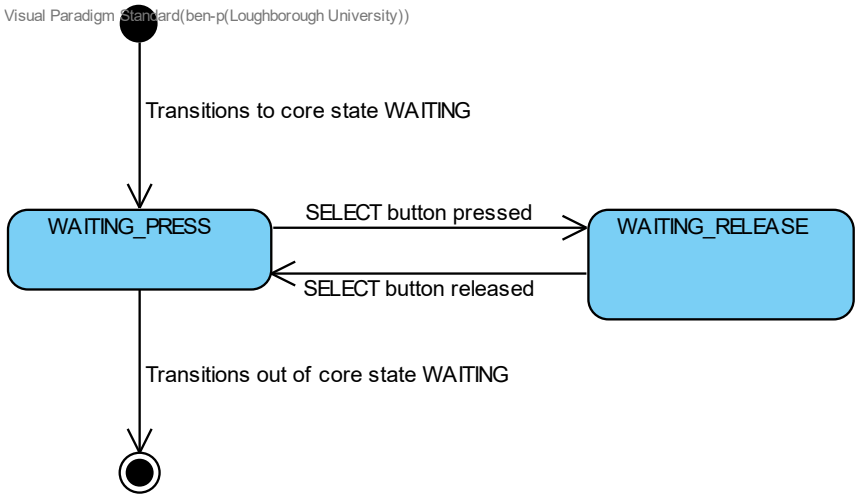
Semester 2

1 FSMs

This is the core finite state machine which I implemented for this project. It uses 7 states, each handling one of the main operations of the system:



This is a sub FSM which I implemented to handle the behaviour of the SELECT button when holding and releasing it to display different screens on the lcd:



2 Data structures

To store the data associated with each channel, I have defined my own structure named "channel" ** *line 17* **, which consists of a char "id", char array "description", int "value", int "min" and int "max". I have used an array of size 26 and type "channel" named "channelArray", to hold each of the 26 possible channels and their data **_line 62_**. This array is sorted using a bubble sort **_line 485_** each time a new channel is added to ensure that all channel data is stored in alphabetical order to simplify other tasks. "channel" has been instantiated as "newChannel" **_line 451_**, which is later appended to "channelArray" if the channel didn't exist prior to the entry of this message.

To handle the FSM functionalities, I have defined two enums: state_e containing "SYNCHRONISATION", "INITIALISATION", "WAITING", "NEW_CHANNEL", "VALUE", "MAX", "MIN" ** *line 58*, which is initialized as "state" **_on line 322_** and state_b containing "WAITING_PRESS" and "WAITING_RELEASE" ** *line 59*, which is initialized as "buttonState" **_on line 365_**.

I didn't create functions to update the global channelArray, however each state updates the array instead, details are below:

NEW_CHANNEL: Either updates the description of the pre-existing channel with the entered channel id, or creates a new entry to the array, adding the channel id and description. Also sorts the array each run through.

VALUE: Updates the stored value in the array for the corresponding channel id. If no such channel has been intialised, it makes no changes.

MAX: Updates the stored max value in the array for the entered corresponding channel id. If no channel with that id has been created, it makes no changes.

MIN: Updates the stored min value in the array for the entered corresponding channel id. If no channel with that id has been created, it makes no changes.

Definition of State Enums: (lines 58-59)

```
enum state_e { SYNCHRONISATION = 3, INITIALISATION, WAITING, NEW_CHANNEL, VALUE, MAX, MIN }; // the main states

enum state_b { WAITING_PRESS = 8, WAITING_RELEASE }; // states for operation of buttons
```

Definition of bubble sort:

(lines 485-493)

```
for (int x = 0; x < channelArrayLength; x++){
  for (int y = 0; y < channelArrayLength - 1; y++){
    if (channelArray[y].id > channelArray[y+1].id){
      channel tempChan = channelArray[y+1];
      channelArray[y+1] = channelArray[y];
      channelArray[y] = tempChan;
    }
  }
}
```

Definition of "channel" Structure:

(lines 17-23)

```
struct channel {
  char id;
  char description[16] = "            ";
  int value = -1;
  int minValue = 0;
  int maxValue = 255;
};
```

3 Debugging

I have used several Serial.println() calls that are not commented out to indicate whether functions are being ran properly and that the correct values are in the correct places in both the array I have used and when displaying to the screen. I have included one example of this below, taken from lines 495-508. This returns the id, description, value, min and max for every channel stored in the array, as well as the length of the array. I would use this example when checking that the inputs had been stored correctly when adding new data for a channel or creating a new channel.

More examples can be found throughout the code.

```
/*
    for (int x = 0; x < channelArrayLength; x++){
     Serial.println("DEBUG: " + (String)channelArray[x].id);
     Serial.print("DEBUG: ");
     for (int y = 0; y<15; y++){
       Serial.print(channelArray[x].description[y]);
     }
     Serial.println();
     Serial.println("DEBUG: " + (String)channelArray[x].value);
     Serial.println("DEBUG: " + (String)channelArray[x].maxValue);
     Serial.println("DEBUG: " + (String)channelArray[x].minValue);
    }
    Serial.println(channelArrayLength);
    */
```

4 Reflection

There are a few small issues which I encountered while completing this project. Firstly, rarely, pressing one of the up/down buttons wouldn't be detected by the Arduino and it would have to be pressed again. I believe this is an error which would occur during the WAITING state whereby if the button was pressed in the 5ms between the "Serial.readString()" call on line 337 and the timeout from this call, the action of pressing it would not be detected by the lcd.readButtons() call on line 370. I believe this issue could potentially be fixed by using an augmented method of operating the buttons, similar to the select button, whereby the code would be scrolled upon changing state to the WAITING_RELEASE buttonState, and then would return to the WAITING_PRESS state upon release of the button. I believe this would work as the SELECT button uses this system and I have had no issues with detecting when it has been pressed.

Additionally, my SCROLL implementation is another area which I feel could be improved. The code that I wrote will scroll a full 15 characters regardless of the length of the actual name/description that is being displayed. I would fix this by making a change to my channel Structure, adding an integer variable "lengthOfName" and store the length of the name from in the array along with the name when creating a new channel. I would then use this value to determine when the end of the name had been reached to reset the scrolling to the start of the name.

One part of my implementation which I quite like is that when the backlight changes colour to reflect a value outside of the range, I added a counter so that after 5 new values have been added, the colour change times out, setting the backlight colour back to white, as it says in documentation that "if all recent values are within range on every channel, then set the backlight to white". My interpretation of this was 5 for demonstration purposes however this feature could be adapted depending on one's interpretation of "recent values".

Overall, I am very happy with my solution.

5 UDCHARS

I have defined up and down arrows to replace ^ and v on my display. The definitions for these are in text boxes below and can be found in the .ino file on the indicated lines.

Definition of up arrow:

(lines 26-35)

```
byte upArrow[8] = {
  B00100,
  B01110,
  B11111,
  B00100,
  B00100,
  B00100,
  B00100,
  B00100,
};
```

Definition of down arrow:

(lines 37-46)

```
byte downArrow[8] = {
  B00100,
  B00100,
  B00100,
  B00100,
  B00100,
  B11111,
  B01110,
  B00100,
};
```

These are used to create characters in the setup() function using the .createChar() function:

Creating UDCHARS: (lines 54-55)

```
lcd.createChar(0, upArrow);
lcd.createChar(1, downArrow);
```

And displayed on the lcd in my atTop(), atBtm(), and inMiddle() functions:

atTop() function:

(lines 92-97)

```
void atTop(){
  lcd.setCursor(0,0);
  lcd.print(' ');
  lcd.setCursor(0,1);
  lcd.write(byte(1));
}
```

atBtm() function

(lines 98-103)

```
void atBtm(){
  lcd.setCursor(0,0);
  lcd.write(byte(0));
  lcd.setCursor(0,1);
  lcd.print(' ');
}
```

inMiddle() function

(lines 104-109)

```
void inMiddle(){
  lcd.setCursor(0,0);
  lcd.write(byte(0));
  lcd.setCursor(0,1);
  lcd.write(byte(1));
}
```

6 FREERAM

I have implemented this extension, copying across the code from lab workshop 3 from lines 73-89 and making changes to my selectDisplay() function, which changes the lcd screen when the select button is pressed, so that the free memory is printed on the line below my student ID number.

selectDisplay() function:

(lines 301-309)

```
void selectDisplay(){ // changes the display when "select" button is pressed
 lcd.setBacklight(5);//set it purple
 lcd.setCursor(0,0);
 lcd.print("F128493        ");
 lcd.setCursor(0,1);
 lcd.print("               ");
 lcd.setCursor(0,1);
 lcd.print("FREE: " + (String)freeMemory());
// outputs the amount of free ram to the select button interface
 }
```

7 HCI

8 EEPROM

9 RECENT

10 NAMES

I implemented this extension, making a few changes to my updateDisplay() function to display the name of each channel. My channel names/descriptions are stored as a char array of length 16 as part of my structure "channel" (I have included the definition of this below for reference). As I have also implemented the scrolling feature, the code I have provided below is only the code for if the name doesn't need to scroll, as this is what I added to the program when completing the NAMES extension. The code for displaying the name while scrolling shall be evidenced in the SCROLL section of this document. I have also included the code which runs prior to this, which displays the channel id and value in the lcd, to explain the lack of cursor positioning in the latter code, as this is not necessary after the previous lines, which naturally position the cursor.

Definition of "channel" Structure:

(lines 17-23)

```
struct channel {
  char id;
  char description[16] = "              ";
  int value = -1;
  int minValue = 0;
  int maxValue = 255;
};
```

Displaying channel id and value

(lines 127-133)

```
lcd.print(channelArray[topDisplay].id);
   if (channelArray[topDisplay].value > -1){
     lcd.print(dispVal);
   }else{
     lcd.print("   ");
   }
   lcd.print(" ");//displaying name
```

Displaying of description without scrolling:

(lines 158-160)

```
for (int x = 0; x < 10; x++){
       lcd.print(channelArray[topDisplay].description[x]);
     }
```

11 SCROLL

I have implemented this extension as well as the NAMES extension, For this extension I did not need to make any changes to the FSM as I only had to change the updateDisplay() function which displays channel id, value and names to the LCD, which is called from the WAITING state.

I defined 2 (one for top lcd row and one for bottom) versions of 3 different static variables for this implementation:

needScroll is used to determine whether a line needs to scroll or not

now is used for the timing of the scrolling. This is initialized as the current time when the code is first run.

scrollCount is keeps track of how many characters have scrolled along so that the scrolling can return to the start at the end of the name.

Definitions of required variables for SCROLL (lines 65-70)

```
static bool needScroll1;
static bool needScroll2;
static unsigned long now1 = millis();
static unsigned long now2 = millis();
static int scrollCount1;
static int scrollCount2;
```

Continues on next page

There are three locations where the code (or a version of it) is used to scroll the name in my implementation: the one evidenced below between lines 133-161, for scrolling one name in the situation that there is only one channel that has been created, on the top line, and that it needs to scroll, one between lines 190-218 for when there are at least two channels created and the top row needs to scroll and one between 228-256 for when there are at least two channels and the bottom row needs to scroll.

Scrolling (lines 133-161)

```
lcd.print(" ");// displaying name
//this will determine whether the name is long enough to require scrolling
   needScroll1 = false;
   for (int y = 10; y < 15; y++){
    if (channelArray[topDisplay].description[y] != ' '){
     needScroll1 = true;
    }
   }
   if (needScroll1 == true){
    //Serial.println("Needs to scroll");
// if it has scrolled along 5, reset to 0
    if(scrollCount1 > 5){
     scrollCount1 = 0;
    }
//write to lcd within range
    for (int x = 0; x < 10; x++){
     lcd.write(channelArray[topDisplay].description[x+scrollCount1]);
    }
//scroll along one every half second
    if (millis() - now1 > 500){
     scrollCount1++;
     now1 = millis();
    }
   }else{
    for (int x = 0; x < 10; x++){
     lcd.print(channelArray[topDisplay].description[x]);
    }
   }
```