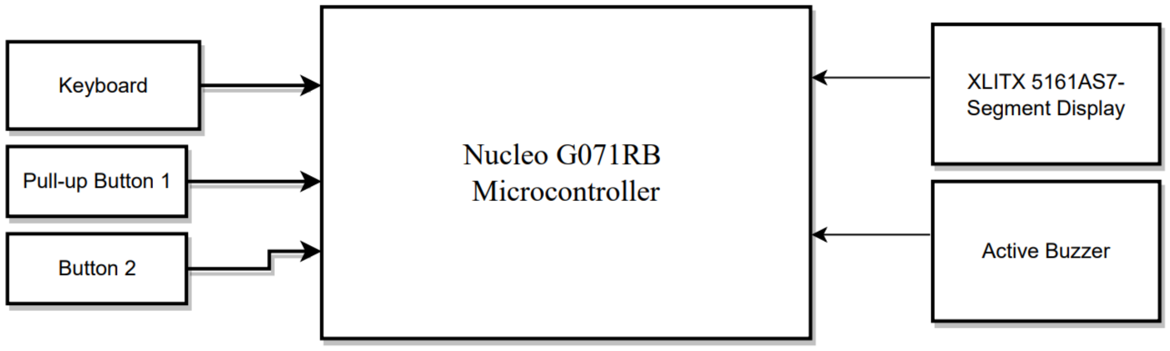
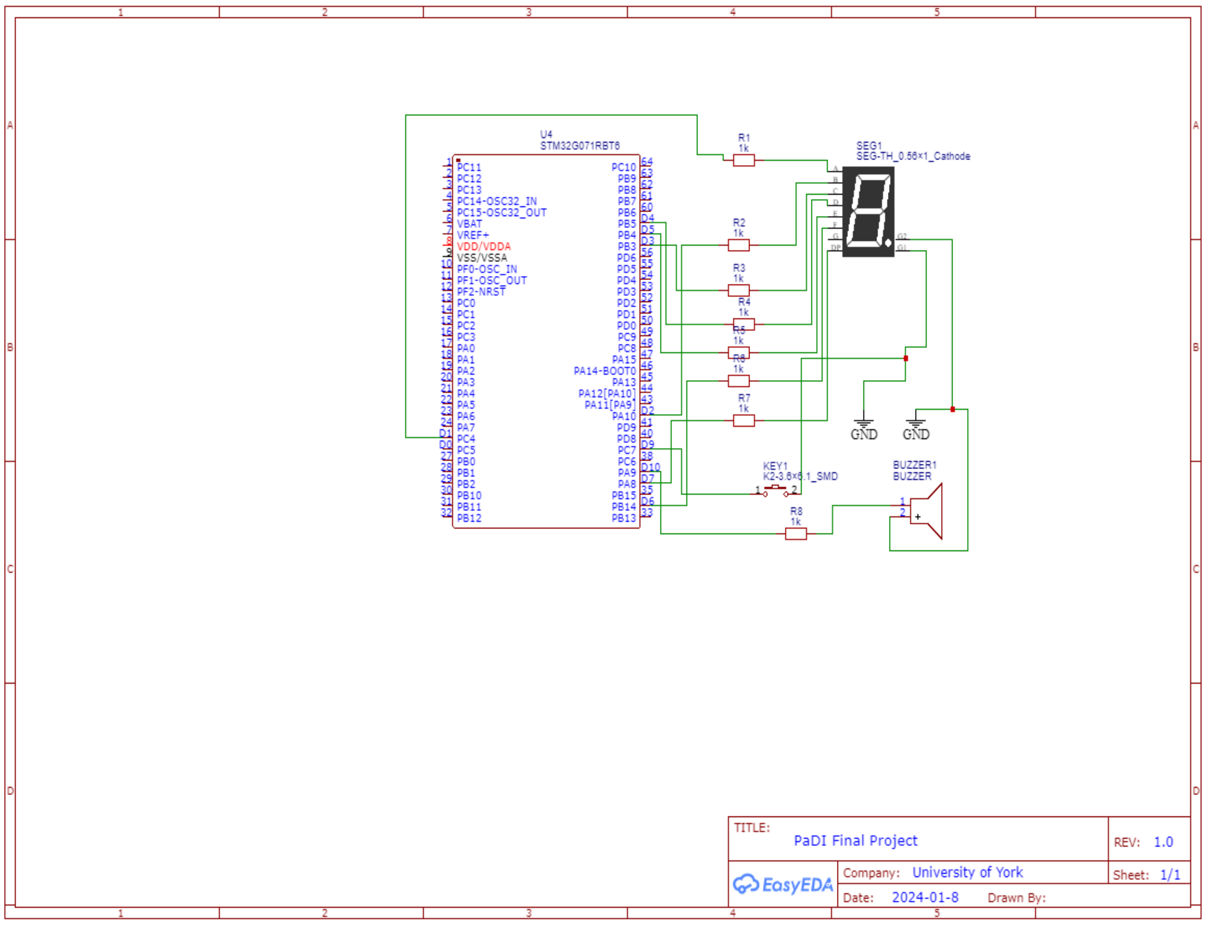
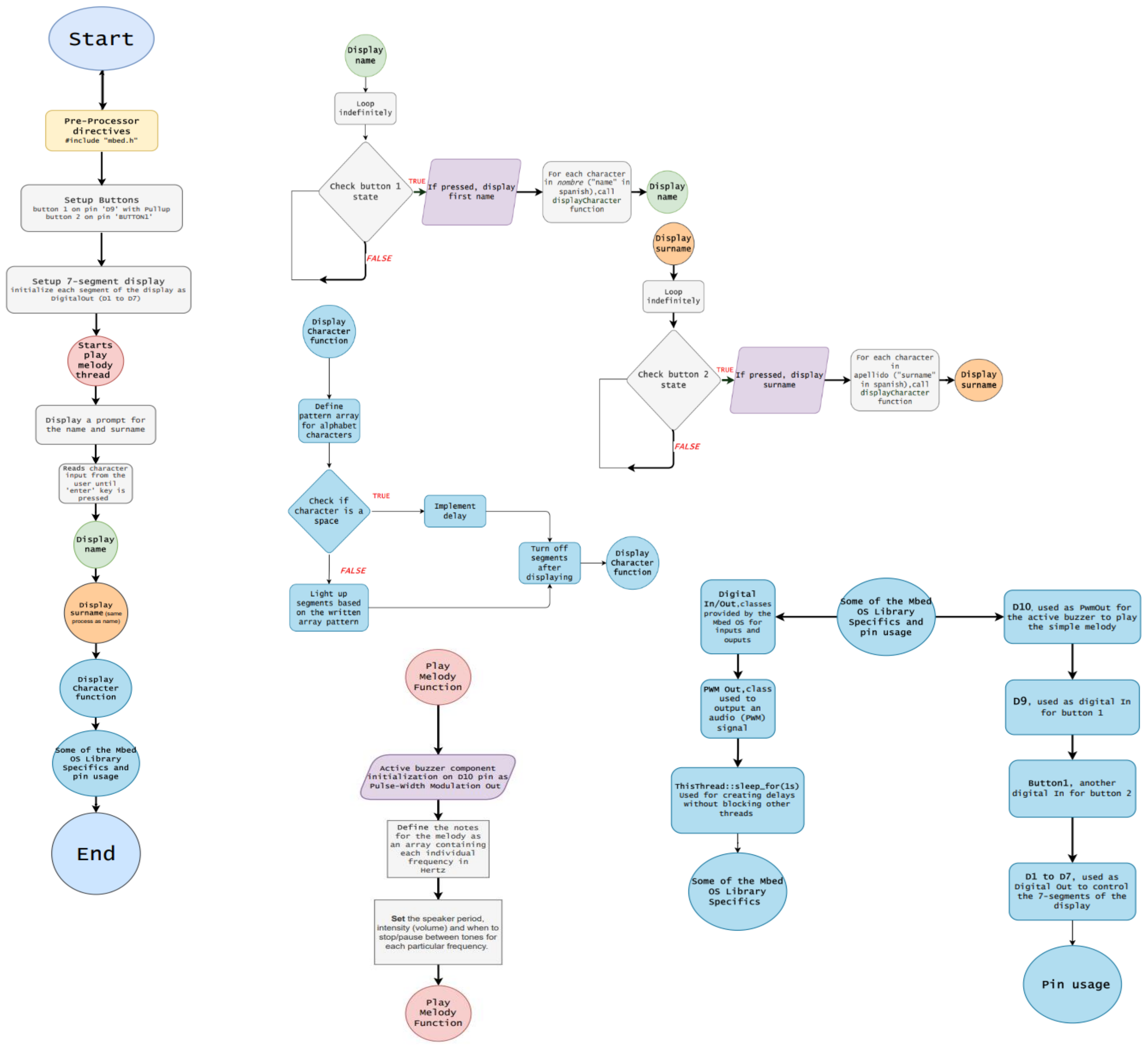
1-Block Diagram



2-Schematic



Flowchart

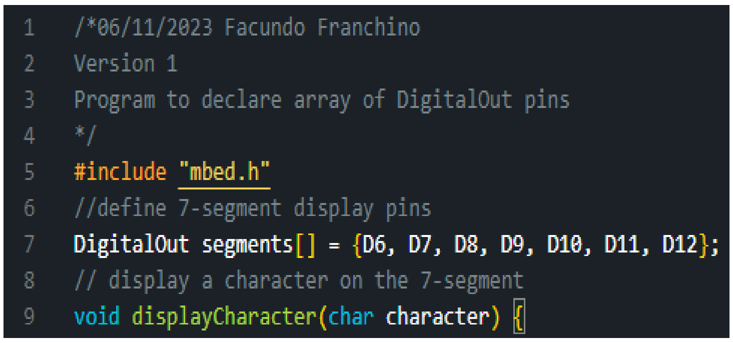

Start
Pre-Processor directives
#include "mbed.h"
Setup Buttons
button 1 on pin 'D9' with Pullup
button 2 on pin 'BUTTON1'
Setup 7-segment display
initialize each segment of the display as DigitalOut (D1 to D7)
Starts play melody thread
Display a prompt for the name and surname
Reads character input from the user until 'enter' key is pressed
Display name
Display surname (same process as name)
Display Character function
Some of the Mbed OS Library Specifics and pin usage
End
Display name
Loop indefinitely
Check button 1 state
TRUE
FALSE
If pressed, display first name
For each character in nombre ("name" in spanish),call displayCharacter function
Display name
Display surname
Loop indefinitely
Check button 2 state
TRUE
FALSE
If pressed, display surname
For each character in apellido ("surname" in spanish),call displayCharacter function
Display surname
Display Character function
Define pattern array for alphabet characters
Check if character is a space
TRUE
FALSE
Implement delay
Turn off segments after displaying
Display Character function
Light up segments based on the written array pattern
Play Melody Function
Active buzzer component initialization on D10 pin as Pulse-Width Modulation Out
Define the notes for the melody as an array containing each individual frequency in Hertz
Set the speaker period, intensity (volume) and when to stop/pause between tones for each particular frequency.
Play Melody Function
Some of the Mbed OS Library Specifics and pin usage
Digital In/Out,classes provided by the Mbed OS for inputs and ouputs
PWM Out,class used to output an audio (PWM) signal
ThisThread::sleep_for(1s) Used for creating delays without blocking other threads
Some of the Mbed OS Library Specifics
D10, used as PwmOut for the active buzzer to play the simple melody
D9, used as digital In for button 1
Button1, another digital In for button 2
D1 to D7, used as Digital Out to control the 7-segments of the display
Pin usage

4.Code Development Story

This document charts the progression of my final project in Programming and Digital Interfacing, detailing the development from basic functionality to the integration of a set of features reflecting my own creative input.

Version 1

1. This is the first idea I had for the program as I was trying to use the concept of arrays in some manner.
2. DigitalOut segments []= is used to define an array of DigitalOut objects, the pins are labelled D6 to D12 on this first version.
3. void displayCharacter(char character) declares a function called displayCharacter that takes a single char parameter named character. This function is in charge of displaying a character on the 7-segment display.

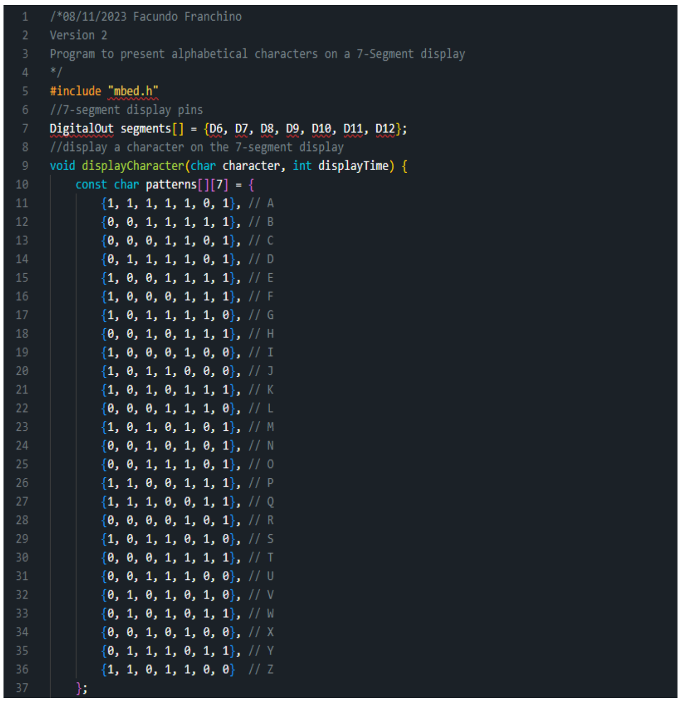


```
1  /*06/11/2023 Facundo Franchino
2  Version 1
3  Program to declare array of DigitalOut pins
4  */
5  #include "mbed.h"
6  //define 7-segment display pins
7  DigitalOut segments[] = {D6, D7, D8, D9, D10, D11, D12};
8  // display a character on the 7-segment
9  void displayCharacter(char character) {
```

Debugging log:

-No particular errors while doing this, besides from a missing ";" in line 7(this will be referenced in the next version)

Version 1$\longrightarrow$Version 2



```
1  /*08/11/2023 Facundo Franchino
2  Version 2
3  Program to present alphabetical characters on a 7-Segment display
4  */
5  #include "mbed.h"
6  //7-segment display pins
7  DigitalOut segments[] = {D6, D7, D8, D9, D10, D11, D12};
8  //display a character on the 7-segment display
9  void displayCharacter(char character, int displayTime) {
10     const char patterns[][7] = {
11         {1, 1, 1, 1, 1, 0, 1}, // A
12         {0, 0, 1, 1, 1, 1, 1}, // B
13         {0, 0, 0, 1, 1, 0, 1}, // C
14         {0, 1, 1, 1, 1, 0, 1}, // D
15         {1, 0, 0, 1, 1, 1, 1}, // E
16         {1, 0, 0, 0, 1, 1, 1}, // F
17         {1, 0, 1, 1, 1, 1, 0}, // G
18         {0, 0, 1, 0, 1, 1, 1}, // H
19         {1, 0, 0, 0, 1, 0, 0}, // I
20         {1, 0, 1, 1, 0, 0, 0}, // J
21         {1, 0, 1, 0, 1, 1, 1}, // K
22         {0, 0, 0, 1, 1, 1, 0}, // L
23         {1, 0, 1, 0, 1, 0, 1}, // M
24         {0, 0, 1, 0, 1, 0, 1}, // N
25         {0, 0, 1, 1, 1, 0, 1}, // O
26         {1, 1, 0, 0, 1, 1, 1}, // P
27         {1, 1, 1, 0, 0, 1, 1}, // Q
28         {0, 0, 0, 0, 1, 0, 1}, // R
29         {1, 0, 1, 1, 0, 1, 0}, // S
30         {0, 0, 0, 1, 1, 1, 1}, // T
31         {0, 0, 1, 1, 1, 0, 0}, // U
32         {0, 1, 0, 1, 0, 1, 0}, // V
33         {0, 1, 0, 1, 0, 1, 1}, // W
34         {0, 0, 1, 0, 1, 0, 0}, // X
35         {0, 1, 1, 1, 0, 1, 1}, // Y
36         {1, 1, 0, 1, 1, 0, 0}  // Z
37     };
```

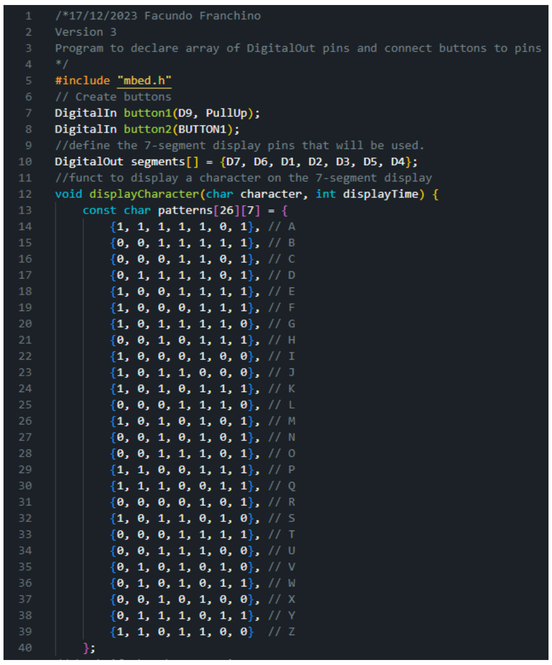
1. It was a test on my ability to program patterns to match those given for 7-segment displays for each letter of the alphabet. To do this, I wrote each of these individually in a text file prior to coding the array.
2. const char patterns [][7]={ defines an array called 'patterns' that stores the specific patterns written to display all 26 letters on the alphabet on the 7-segment display. A key thing to note here, const indicates these values inside the array won't be modified.
3. Lines 12 to 37 represent different binary combinations to represent characters A-Z on the 7-segment display. Binary (1s and 0s) is used as either one or the other indicates whether a specific segment should be on (1=True) or off (0=True). The comments on each of these letters make it easy for the reader to know which letter the pattern in question represents.

Debugging log:

-Mistakes in the written pattern for letters ‘O’ and ‘S’ appeared while testing patterns for the whole alphabet.

-Missing “;”in line 7 carried from the first version lead to failed build.

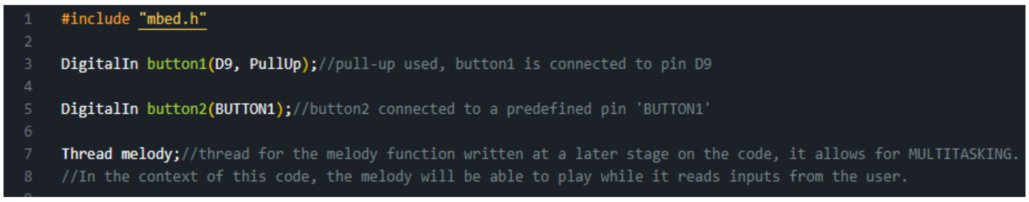
Version 2→Version 3



```
1   /*17/12/2023 Facundo Franchino
2   Version 3
3   Program to declare array of DigitalOut pins and connect buttons to pins
4   */
5   #include "mbed.h"
6   // Create buttons
7   DigitalIn button1(D9, PullUp);
8   DigitalIn button2(BUTTON1);
9   //define the 7-segment display pins that will be used.
10  DigitalOut segments[] = {D7, D6, D1, D2, D3, D5, D4};
11  //funct to display a character on the 7-segment display
12  void displayCharacter(char character, int displayTime) {
13      const char patterns[26][7] = {
14          {1, 1, 1, 1, 1, 0, 1}, // A
15          {0, 0, 1, 1, 1, 1, 1}, // B
16          {0, 0, 0, 1, 1, 0, 1}, // C
17          {0, 1, 1, 1, 1, 0, 1}, // D
18          {1, 0, 0, 1, 1, 1, 1}, // E
19          {1, 0, 0, 0, 1, 1, 1}, // F
20          {1, 0, 1, 1, 1, 1, 0}, // G
21          {0, 0, 1, 0, 1, 1, 1}, // H
22          {1, 0, 0, 0, 1, 0, 0}, // I
23          {1, 0, 1, 1, 0, 0, 0}, // J
24          {1, 0, 1, 0, 1, 1, 1}, // K
25          {0, 0, 0, 1, 1, 1, 0}, // L
26          {1, 0, 1, 0, 1, 0, 1}, // M
27          {0, 0, 1, 0, 1, 0, 1}, // N
28          {0, 0, 1, 1, 1, 0, 1}, // O
29          {1, 1, 0, 0, 1, 1, 1}, // P
30          {1, 1, 1, 0, 0, 1, 1}, // Q
31          {0, 0, 0, 0, 1, 0, 1}, // R
32          {1, 0, 1, 1, 0, 1, 0}, // S
33          {0, 0, 0, 1, 1, 1, 1}, // T
34          {0, 0, 1, 1, 1, 0, 0}, // U
35          {0, 1, 0, 1, 0, 1, 0}, // V
36          {0, 1, 0, 1, 0, 1, 1}, // W
37          {0, 0, 1, 0, 1, 0, 0}, // X
38          {0, 1, 1, 1, 0, 1, 1}, // Y
39          {1, 1, 0, 1, 1, 0, 0}  // Z
40      };
```

1. DigitalIn button1(D9,PullUp); initializes an input pin with a pull-up for a button on the nucleo connected to pin D9.
2. DigitalIn button1(BUTTON1); initializes a second input pin for a specific button, using a pin named BUTTON1.
3. There’s a change in line 12 when defining the function, I added int DisplayTime, to determine how long a character will be displayed for.
4. Another change was done in line 13, where I specified 26 rows and 7 columns in regards to the amount of letters on the alphabet. Previously, the amount of rows was undefined.

Version 3→Version 4



```
#include "mbed.h"

DigitalIn button1(D9, PullUp);//pull-up used, button1 is connected to pin D9

DigitalIn button2(BUTTON1);//button2 connected to a predefined pin 'BUTTON1'

Thread melody;//thread for the melody function written at a later stage on the code, it allows for MULTITASKING.
//In the context of this code, the melody will be able to play while it reads inputs from the user.
```

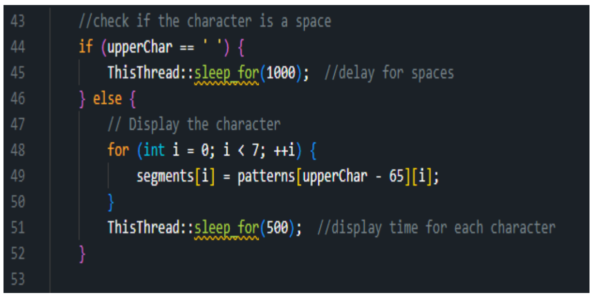
This fourth version was done at a later stage, but played a pivotal role in the program’s ability to multitask. This will make more sense after going through the other functions… Though, in a nutshell, the multitasking I’m addressing is between a melody that’s played while the program reads inputs from the user. This was made possible by using the concept of Threads.

11/11/2023 -12/11/2023

Now onto the next part of the code. This bit took quite a while to figure out, but it’s a good example of applying previous programming concept knowledge into a new language. About three years ago I started learning Python and C, and one of the first concepts I came across was ASCII values. After a few days of trying out non-efficient or functional solutions to assign binary combinations defined in the array to characters, I recalled that ASCII could come in extremely handy for the purpose of alphanumeric conversion. Since ‘A’ is represented by 65, ‘B’ by 66 and so on, I figured the ASCII values of characters could be assigned in order (65 to 91 ‘A-Z’) to the previous array of patterns created (0 to 26 'A-Z'). In short, by converting characters to ASCII and then subtracting its respective numeric representation in the table, I would get a matching index to look up in the array of patterns.

It's worth noting that in my failed first attempt to carry out this idea I tried converting the character to uppercase manually. Though I didn't get it to work at the time. Hence, this is referenced in the figure below as 'upperChar':

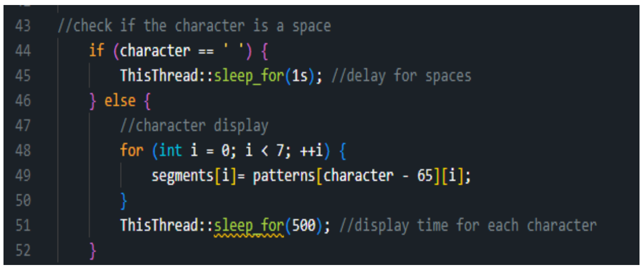
Version 1→



```
//check if the character is a space
if (upperChar == ' ') {
    ThisThread::sleep_for(1000);  //delay for spaces
} else {
    // Display the character
    for (int i = 0; i < 7; ++i) {
        segments[i] = patterns[upperChar - 65][i];
    }
    ThisThread::sleep_for(500);  //display time for each character
}
```

The next day, I realised it wasn't even necessary to convert to uppercase manually in the first place, and that I could just use the char character variable declared on the function. This is how it appeared with the new implementation.

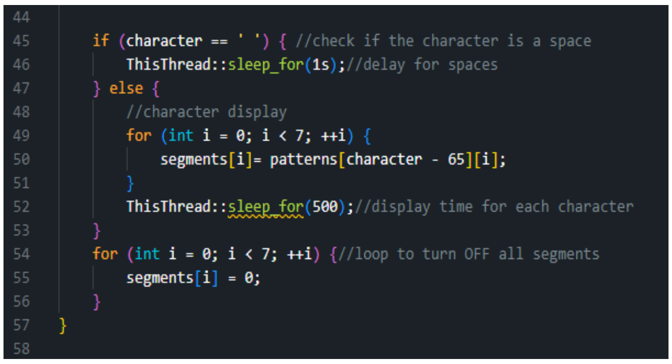
Version1→Version2



```
//check if the character is a space
    if (character == ' ') {
        ThisThread::sleep_for(1s); //delay for spaces
    } else {
        //character display
        for (int i = 0; i < 7; ++i) {
            segments[i]= patterns[character - 65][i];
        }
        ThisThread::sleep_for(500); //display time for each character
    }
```

Ultimately, I implemented a loop to turn off all segments on the 7-segment display. Here's its integration with earlier versions:

Version2→Version3

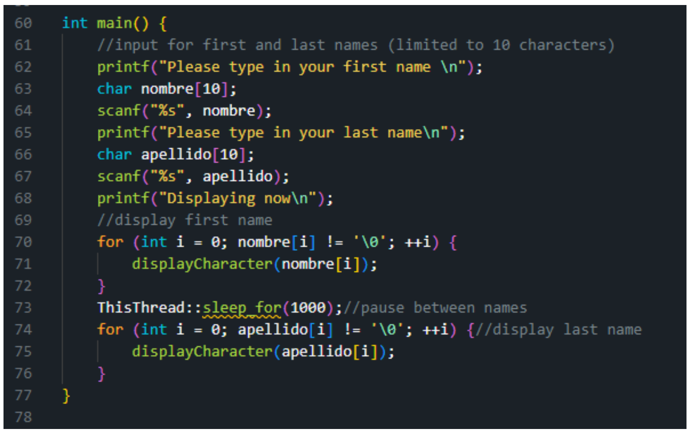


```
        if (character == ' ') { //check if the character is a space
            ThisThread::sleep_for(1s);//delay for spaces
        } else {
            //character display
            for (int i = 0; i < 7; ++i) {
                segments[i]= patterns[character - 65][i];
            }
            ThisThread::sleep_for(500);//display time for each character
        }
        for (int i = 0; i < 7; ++i) {//loop to turn OFF all segments
            segments[i] = 0;
        }
    }
```

14/11/2023

Next, I started working on the main function so as to be able to test the basic functionality for the first time. This function evolved quite a lot with time, but the first version done back in said date looked like this:

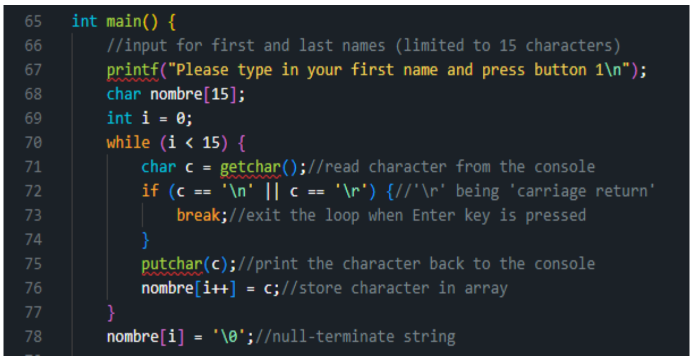
Version 1→



```c
int main() {
    //input for first and last names (limited to 10 characters)
    printf("Please type in your first name \n");
    char nombre[10];
    scanf("%s", nombre);
    printf("Please type in your last name\n");
    char apellido[10];
    scanf("%s", apellido);
    printf("Displaying now\n");
    //display first name
    for (int i = 0; nombre[i] != '\0'; ++i) {
        displayCharacter(nombre[i]);
    }
    ThisThread::sleep_for(1000);//pause between names
    for (int i = 0; apellido[i] != '\0'; ++i) {//display last name
        displayCharacter(apellido[i]);
    }
}
```

Apart from a few syntax errors I recall fixing on the first few builds, it worked. This meant the basic functionality of the project was almost done. Though, I wasn't quite satisfied with this version and tried to build up more on it.

Version1→Version2

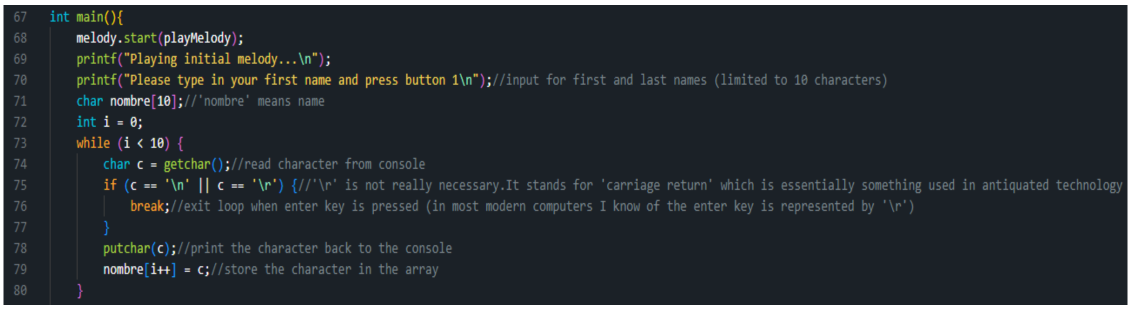


```c
int main() {
    //input for first and last names (limited to 15 characters)
    printf("Please type in your first name and press button 1\n");
    char nombre[15];
    int i = 0;
    while (i < 15) {
        char c = getchar();//read character from the console
        if (c == '\n' || c == '\r') {//'\r' being 'carriage return'
            break;//exit the loop when Enter key is pressed
        }
        putchar(c);//print the character back to the console
        nombre[i++] = c;//store character in array
    }
    nombre[i] = '\0';//null-terminate string
```

On this new version, I decided it could be simpler to read character by character with getchar rather than an entire string at once with scanf. As I said, this was an approach for more simplicity, both versions work just fine but I like reading from console with getchar and printing the character back to console with putchar better.

16/11/2023

Version3→Version4

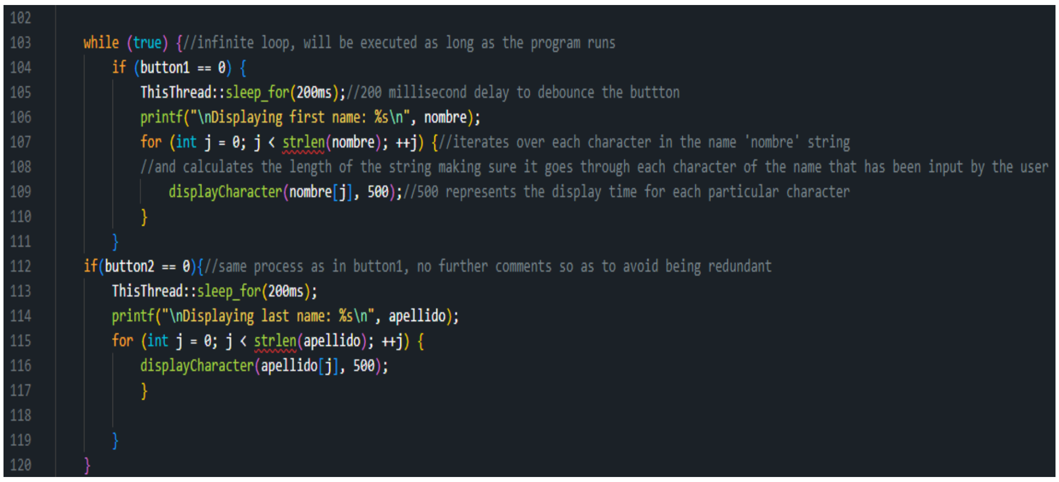


```c
67  int main(){
68      melody.start(playMelody);
69      printf("Playing initial melody...\n");
70      printf("Please type in your first name and press button 1\n");//input for first and last names (limited to 10 characters)
71      char nombre[10];//'nombre' means name
72      int i = 0;
73      while (i < 10) {
74          char c = getchar();//read character from console
75          if (c == '\n' || c == '\r') {//'\r' is not really necessary.It stands for 'carriage return' which is essentially something used in antiquated technology
76              break;//exit loop when enter key is pressed (in most modern computers I know of the enter key is represented by '\r')
77          }
78          putchar(c);//print the character back to the console
79          nombre[i++] = c;//store the character in the array
80      }
```

On this fourth version, which also came at a later stage in the project, I changed the character limit to 10 since 15 seemed a bit pointless. We can also see the thread feature mentioned earlier implemented on line 68, which is the instance where the thread is initiating its execution while passing the 'melody' object as an argument.

After testing it out and confirming that it was working fine, I did the exact same thing for surname. In addition, I added an infinite loop with a series of ‘if’ statements to check if both buttons were being pressed. The exact same process was carried out for both as shown on the following screenshot:

Version3→Version4



```
    while (true) {//infinite loop, will be executed as long as the program runs
        if (button1 == 0) {
            ThisThread::sleep_for(200ms);//200 millisecond delay to debounce the buttton
            printf("\nDisplaying first name: %s\n", nombre);
            for (int j = 0; j < strlen(nombre); ++j) {//iterates over each character in the name 'nombre' string
            //and calculates the length of the string making sure it goes through each character of the name that has been input by the user
                displayCharacter(nombre[j], 500);//500 represents the display time for each particular character
            }
        }
    if(button2 == 0){//same process as in button1, no further comments so as to avoid being redundant
        ThisThread::sleep_for(200ms);
        printf("\nDisplaying last name: %s\n", apellido);
        for (int j = 0; j < strlen(apellido); ++j) {
            displayCharacter(apellido[j], 500);
        }

        }
    }
```

20/11/23 to 26/11/23

From the outset of this project, I was drawn to the idea of adding a musical dimension. After listening to the Windows 95 startup sound one morning, I figured that a simple melody that could play simultaneously with the user’s surname and name inputs could be added. The prospect of developing this feature, especially in an unfamiliar environment like mBed, seemed like an exciting challenge.

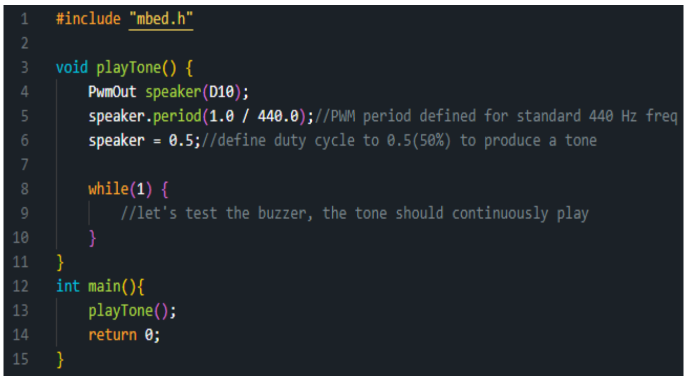
Initially, I was completely lost on how to approach the idea. Also, I knew there were hardware limitations as the active buzzer can only play single tones. Although this wasn’t absolutely satisfactory, I figured the best way to try it out was by loading a sequence of pitches into an array. But, how would the program interpret those frequency figures and turn them into sound? Well, that was the tough part to figure out.

During my investigation to make this idea work, I came through the concept of PWM, this is a technique to manipulate analogue devices using digital signals. The way in which they relate is essentially by setting the period of the PWM signal. I knew from my maths studies that the relationship between frequency (f) and period (T) is the following:

$$f = 1/T$$

So, I figured that if I managed to set the period of the signal, I could thence set the frequency of a soundwave. Creating a tone would essentially be composed of setting a PWM signal to a particular period and note frequency, since the speaker will vibrate at that frequency producing a soundwave that humans perceive as tone! This was a huge turning point for me, now what was left to do was to figure out how to actually turn this idea into code.

Version1→



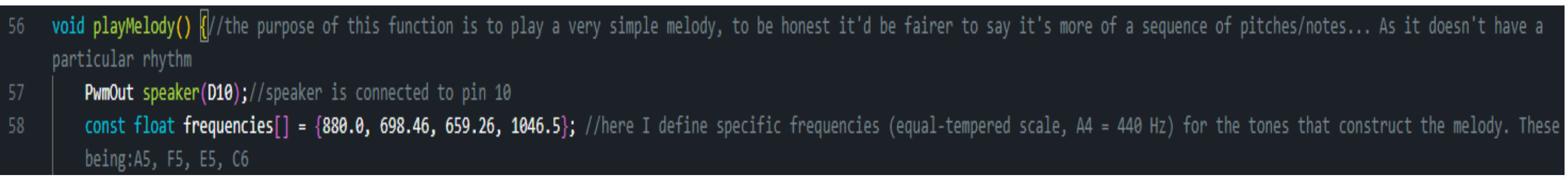
```
#include "mbed.h"

void playTone() {
    PwmOut speaker(D10);
    speaker.period(1.0 / 440.0);//PWM period defined for standard 440 Hz freq
    speaker = 0.5;//define duty cycle to 0.5(50%) to produce a tone

    while(1) {
        //let's test the buzzer, the tone should continuously play
    }
}
int main(){
    playTone();
    return 0;
}
```

This is a simple separate program I wrote with the aim of playing a standard A4 tone (440hz) continuously. In this, there’s another concept I learned about called duty cycle (DC). Having DC at 0.5 means that the signal is on only half of the time. When used with a buzzer, DC affects the intensity of the signal (volume); lower DC's mean lower sounds due to the fact that the piezoelectric element spends less time in a vibrational state.

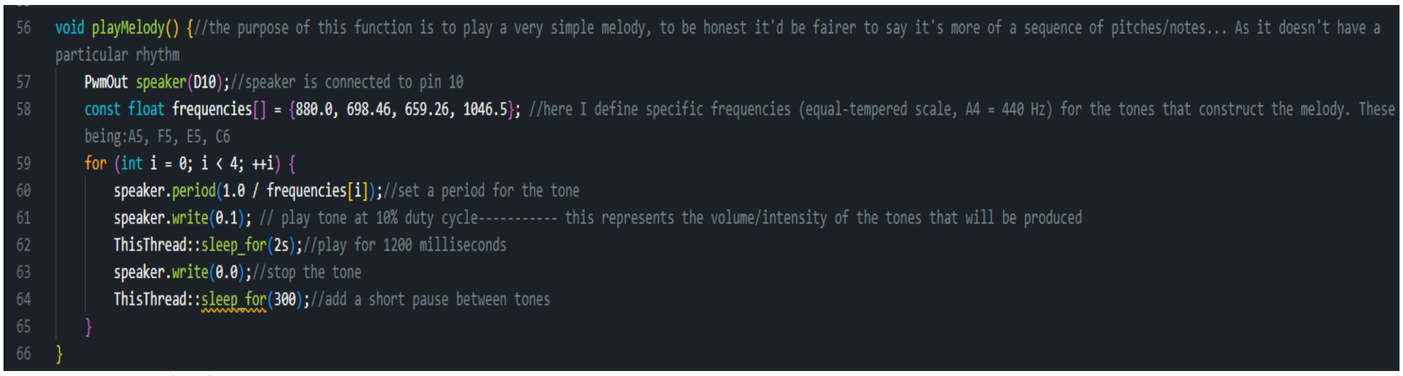
Version1→Version2



```
56 void playMelody() {//the purpose of this function is to play a very simple melody, to be honest it'd be fairer to say it's more of a sequence of pitches/notes... As it doesn't have a particular rhythm
57     PwmOut speaker(D10);//speaker is connected to pin 10
58     const float frequencies[] = {880.0, 698.46, 659.26, 1046.5}; //here I define specific frequencies (equal-tempered scale, A4 = 440 Hz) for the tones that construct the melody. These being:A5, F5, E5, C6
```

Here, I put an idea I mentioned earlier into practice. This is loading a set of frequencies, in this case an Fmaj7 chord arpeggio, into an array. I defined it as const float given that not all frequencies in the sequence could be represented as an int; const since I want it to be unchangeable.

Version2→Version3



```
56 void playMelody() {//the purpose of this function is to play a very simple melody, to be honest it'd be fairer to say it's more of a sequence of pitches/notes... As it doesn't have a particular rhythm
57     PwmOut speaker(D10);//speaker is connected to pin 10
58     const float frequencies[] = {880.0, 698.46, 659.26, 1046.5}; //here I define specific frequencies (equal-tempered scale, A4 = 440 Hz) for the tones that construct the melody. These being:A5, F5, E5, C6
59     for (int i = 0; i < 4; ++i) {
60         speaker.period(1.0 / frequencies[i]);//set a period for the tone
61         speaker.write(0.1); // play tone at 10% duty cycle---------- this represents the volume/intensity of the tones that will be produced
62         ThisThread::sleep_for(2s);//play for 1200 milliseconds
63         speaker.write(0.0);//stop the tone
64         ThisThread::sleep_for(300);//add a short pause between tones
65     }
66 }
```

<u>28/11/23</u>

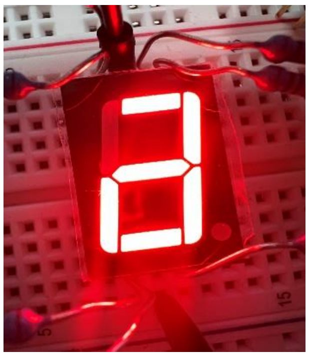
This is the final version of the function, I added a 'for' loop to iterate through the musical note sequence. the condition 'i' runs as long as it's < 4 (number of notes on the array). On line 60 and 61 the PWM period and DC values explained earlier are set. Then I added two of ThisThread::sleep_for lines, one to pause the thread for 2 seconds allowing the tone to play for that duration, and another one to add a short pause (300ms) between tones.

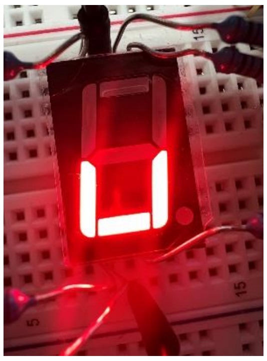
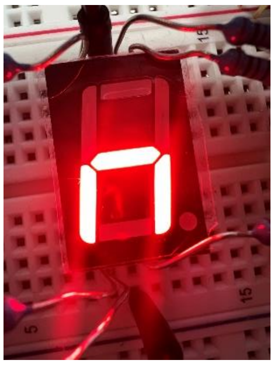
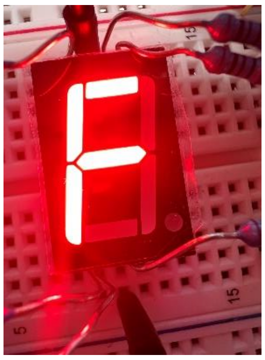
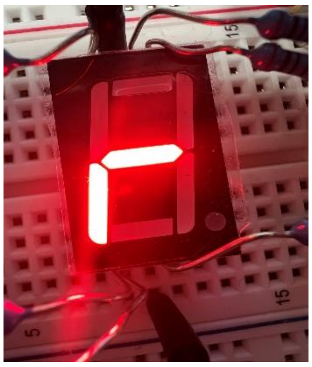
Word count: 1486

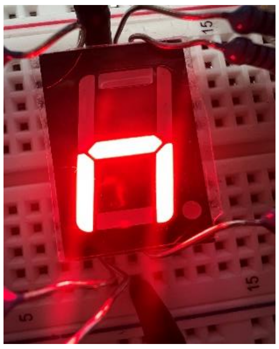
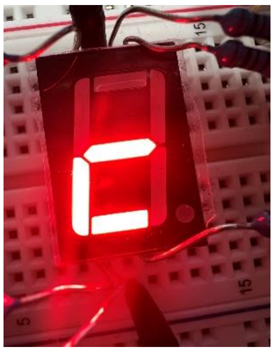
User Guide

- Plug your Nucleo board USB cable into your personal computer
- Store the 'Final_Project_v6.NUCLEO_G071RB.bin' file into the Nucleo folder
- Open a terminal window (I used CoolTerm and set the ports to the max COM available)
- Connect it to the Nucleo board on the terminal so as to have Nucleo inputting information to the written program
- Press the onboard button of the Nucleo ('button2' associated with 'BUTTON1') to start the melody
- Type in your name while the sequence of musical notes plays in the background
- Press the Nucleo button 1 to display the first 5 letters of your first name
- Press the Nucleo button 2 to display the first 5 letter of your surname

Functionality Pictures



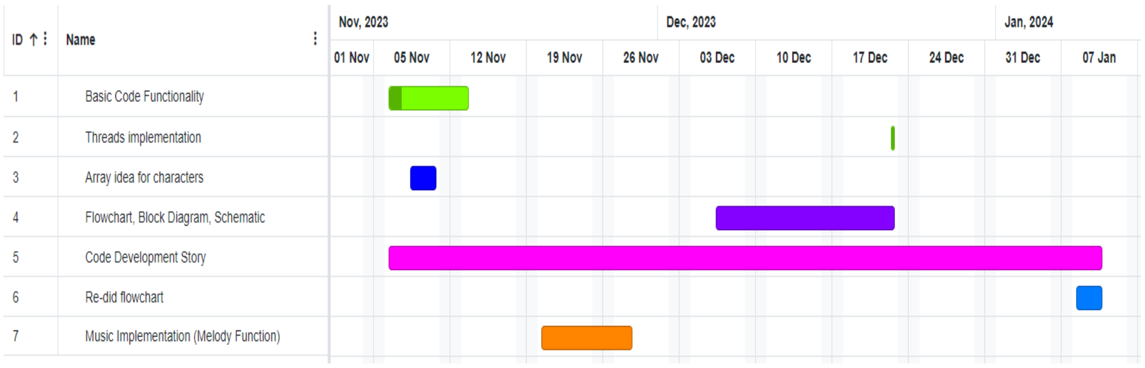





5- Proof of Functionality

link to functionality proof video

Gantt Diagram



ID	Name
1	Basic Code Functionality
2	Threads implementation
3	Array idea for characters
4	Flowchart, Block Diagram, Schematic
5	Code Development Story
6	Re-did flowchart
7	Music Implementation (Melody Function)

Lab Sign-offs

-I have ALL 8 laboratory sign-offs completed which means 100 marks of the 15% final contribution to the module mark.