

# 829H1 Real-Time Embedded Systems Final Report

Candidate No: 105936

12th January 2019

# Contents

<b>1. Abstract</b>	<b>3</b>
<b>2. Introduction</b>	<b>3</b>
<b>3. Exercises</b>	<b>3</b>
3.1. Parallel IO . . . . .	3
3.2. Interrupt and Timers . . . . .	3
3.3. Power Width Modulation . . . . .	4
3.4. Serial Communications . . . . .	4
3.5. Ethernet Connections . . . . .	4
<b>4. Project</b>	<b>4</b>
4.1. Getting the current public IP Address . . . . .	4
4.2. Displaying on the LCD . . . . .	5
4.3. Re-factoring the code . . . . .	5
4.4. Using the joystick to scroll through the display . . . . .	5
4.5. Getting the Location . . . . .	6
4.6. Displaying the Weather Information . . . . .	6
<b>5. Analysis</b>	<b>6</b>
5.1. Program Information/User Guide . . . . .	6
5.2. Review . . . . .	6
<b>6. Conclusion</b>	<b>7</b>
<b>Appendices</b>	<b>8</b>
<b>A. Exercises Code</b>	<b>8</b>
A.1. Write String Timer . . . . .	8
<b>B. Program Extracts</b>	<b>8</b>
B.1. Get IP address via HTTP code . . . . .	8
<b>References</b>	<b>9</b>

# 1. Abstract

The IoT space is full of applications and devices to make life easier this project has also been inspired by this trend. The main section of this report is how to connect the freedom K64F[1] to an external API and display text on the application shield. It looks into the process of displaying current weather information on the application shields display.

# 2. Introduction

This report focuses on using the FRDM-K64F[1] as a connected device to display weather information on the application shields display. Developing for real time systems relies on making sure that there is no delay or wait for the program in terms of using interrupts correctly and making sure the program runs quickly. I will start by going through the exercises I completed during the lab sessions before going on to how I created the weather application which runs on the board.

# 3. Exercises

I conducted a number of exercises/experiments over the course of the term and learnt a number of ways in which the outputs of the board can be used and other programming constructs. In this section I will go over what was learnt and how it helped me with the project which was completed.

## 3.1. Parallel IO

This was a relatively gentle introduction on programming for embedded systems. It allowed me to get used to how to use the mbed IDE and how developing for the board works. This exercise was based on direct programming where the binaries are compiled to run on the specific hardware. This is slightly different from my background in programming for computers where a program is compiled for an Operating system or some other intermediate interface between the program and the hardware.

## 3.2. Interrupt and Timers

Although I did not use the timer functionality in the end for my project the interrupts were very useful and vital in my view for real time embedded systems as it is necessary for systems to be able to manage changes and be able to react to important information. I ended up using the interrupts feature to deal with the inputs from the joystick although I had to go further and use mbed events[2] as well so that I was able to output to the various displays. This is because it is not possible to output to a serial communication from an interrupt. table 1 is an example of some of the results I found while working on the experiments in this section. This experiment was using timers to measure how long it took to write a string to the screen depending on its length. The code for this experiment can be found in section A.1.

Character Length	String	Time taken to print to screen	Difference to one fewer character length
0		0.000019	
1	a	0.002075	0.002056
2	aa	0.003114	0.001039
3	aaa	0.004154	0.00104
4	aaaa	0.005194	0.00104
5	aaaaa	0.006235	0.001041
6	aaaaaa	0.007275	0.00104
7	aaaaaaa	0.008314	0.001039
8	aaaaaaaa	0.009355	0.001041
9	aaaaaaaaa	0.010395	0.00104
10	aaaaaaaaaa	0.011435	0.00104

Table 1: Showing the time taken to print a string to the screen compared to the character length.

### 3.3. Power Width Modulation

This exercise was not particularly useful when building my project however it does provide a good introduction on how motor and other devices work these provide a useful introduction on how to use the oscilloscope and work that would be completed in the next section.

### 3.4. Serial Communications

This helped cement knowledge about how serial communications work while also providing a bit of background information on parallel communication. This was one of the more complicated exercise although did allow for the learning of a number of parts of serial communications such as what the different lines are used for. For example the clock line, MISO, MOSI. I was also able to learn about the different clock modes which would change when the data bits were sent compared to the clock line values. Although the usb mouse functionality was relatively interesting and provided a small insight on the number of features which are available on the board. Figure 1 shows how each signal is interpreted when the byte is transmitted.

### 3.5. Ethernet Connections

This exercise taught me the basics about network communications and provided some of the basic idea for what I could do for my project. Also the IBM application clearly showed what sensors are available to be used.

## 4. Project

The main plan for the project was to create a program which would make the device a simple to use weather station of sorts. It would get its current location and then query a external API for the current weather conditions before displaying the information on the screen.

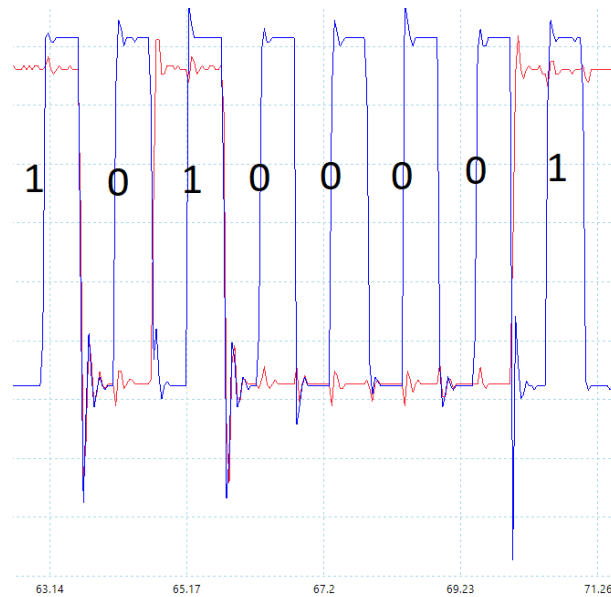


Figure 1: An Image showing the output on the oscilloscope when outputting the byte 10100001

#### 4.1. Getting the current public IP Address

The first part of the program was to get the board to get it's own ip address allowing the board to query the location for the correct IP. To do this I needed to get the current public IP address of the board using the code `EthernetInterface.getIPAddress()` unfortunately this only returned the internal IP address. Therefore, I had to use a public API to get the IP address for this I used <https://api.ipify.org>[3]. To use this I would send off a http request to the address and then read the received text back. To send the http request I had to use the <https://os.mbed.com/teams/sandbox/code/mbed-http/> library simply importing this library didn't work therefore I had to fork the http examples repository found at <https://os.mbed.com/teams/sandbox/code/http-example/> and modify it for my purposes.

#### 4.2. Displaying on the LCD

This was fairly simple to implement compared to the http library. As all I needed to do was import the library from <https://os.mbed.com/users/chris/code/C12832/> and use the example from <https://os.mbed.com/users/chris/code/app-shield-LCD/file/f8ef5e45e488/main.cpp/> as guidance on how to use the library.

#### 4.3. Re-factoring the code

After implementing the function to get the IP address and the display it to the screen I realised I would either have to change the layout of the code so that I could support the display of all of the information or just display one piece of information. I decided that the best way to go was to support displaying all of the information this meant that I would need to create a number of functions to manage what would be displayed and how to change the information on the display.

## 4.4. Using the joystick to scroll through the display

To make use of the joystick I only had to add `InterruptIn joy_up(A2);` for up and down and then make use of the inputs for this. My initial plan was to use the interrupt to update the screens display however it is not possible to call serial communications from an interrupt meaning I cannot use the `printf` to the LCD or to the debug USB. To overcome this issue I need to use mbed events to do this I used information from the blog <https://os.mbed.com/blog/entry/Simplify-your-code-with-mbed-events/>[2]. This would allow me to add functions to be called on a separate thread. After this I now had the basic layout of the system working so all I needed to do was to add functionality to get the location and then weather information.

## 4.5. Getting the Location

For this all I had to do was use the code from earlier to get the IP address but change the URL to an API which would get the location. I also needed to parse the result so that I could make use of the individual JSON values(I.E. the Latitude and Longitude values). to do this I used this library <https://os.mbed.com/users/samux/code/MbedJSONValue/> as it had some documentation on how to use it. This allowed me to relatively easily obtain the latitude and longitude as strings.

## 4.6. Displaying the Weather Information

After I had the obtained the coordinates All I needed to do was to use the API from <https://openweathermap.org/api> to get the weather information for the coordinates I obtained earlier. As I was able to display the temperature fairly quickly I decided on adding a description of the weather to the display. The only slightly complicated thing was converting the temperature value from a double to a string to help me with this I searched stack overflow and found this result <https://stackoverflow.com/a/332132>[4].

# 5. Analysis

## 5.1. Program Information/User Guide

The idea of this program is that all you have to do is plug it in and it works the rest out for you. Therefore to get it working all you need to do is connect the USB and Ethernet and then wait for it to load information. You are able to scroll through the information using the joystick before it loads but all it will read is waiting... but it will refresh once the data is obtained.

## 5.2. Review

I've used the device a couple of time since creating it and it is relatively useful as it is quick to boot up and simply displays the temperature to an accurate level. There are a few improvements which can be made such as remembering the last option selected therefore when the device is connected it does not resort to the ip address display. Another would be that it takes quite a long time to get the IP address. I'm not sure if that is because it is the first http call therefore there is some setup which must be done or the API is a bit slow but it would probably be worth looking into using other API's to get the IP

address to see if they are quicker. Another improvement would be to make the values update every so often or add the functionality so that clicking centre on the joystick would update the values as the only way to do it currently is to reboot the device. One final improvement which could be made is the device could display an icon depending on the weather conditions.

## **6. Conclusion**

Over the course of the module I've learnt a fair amount about developing for embedded systems although I feel that there is much more which can be learnt. I feel that the project I've created is of usable quality although there are some improvements which can be made and have been mentioned in section 5.2. I feel I completed the project I set out to create therefore this has been a success.

# Appendices

## A. Exercises Code

### A.1. Write String Timer

```
1 /* Program: A simple Timer Activate Tera Term terminal to test.*/
2 #include "mbed.h"
3 #include <string>
4
5 Timer t; // define Timer with name "t"
6 Serial pc(USBTX, USBRX);
7 int main()
8 {
9     for(int i = 0; i <= 10; i++) { // Test 1 to 10 characters
10         string str = "";
11         for(int j = 0; j < i; j++) { // Add i characters to string str
12             str += "a";
13         }
14         t.reset();
15         t.start(); //start the timer
16         pc.printf("%s\n", str.c_str());
17         t.stop(); //stop the timer
18         pc.printf("The time taken for %i characters was %f seconds\n", i, t
19             .read()); //print to pc
20     }
```

## B. Program Extracts

### B.1. Get IP address via HTTP code

```
1 char* get_ip_address()
2 {
3
4     HttpRequest* request = new HttpRequest(network, HTTP_GET, "http://api.
5     ipify.org/");
6     request->set_header("Content-Type", "application/json");
7     HttpResponse* response = request->send(body, strlen(body));
8     // if response is NULL, check response->get_error()
9
10    pc.printf("status is %d - %s\n", response->get_status_code(), response
11    ->get_status_message());
12    char* ipAddress = response->get_body_as_string().c_str();
13    pc.printf("body is:\n%s\n", ipAddress);
14
15    delete request; // also clears out the response
16    return ipAddress;
17 }
```

Update  
these



## References

- [1] N. Products. (Mar. 2014). Frdm-k64f: Freedom development platform for kinetis® k64, k63, and k24 mcus, [Online]. Available: <https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/kinetis-cortex-m-mcus/k-seriesperformancem4/k2x-usb/freedom-development-platform-for-kinetis-k64-k63-and-k24-mcus:FRDM-K64F>.
- [2] J. Jongboom. (Jan. 16, 2018). Simplify your code with mbed-events, Arm Limited, [Online]. Available: <https://os.mbed.com/blog/entry/Simplify-your-code-with-mbed-events/> (visited on 01/05/2019).
- [3] R. Degges. (). Ipify, [Online]. Available: <https://www.ipify.org/>.
- [4] J. Schaub, *How do i convert a double into a string in c++?* Dec. 1, 2008. [Online]. Available: <https://stackoverflow.com/a/332132> (visited on 01/07/2019).