829H1 Real-Time Embedded Systems Final Report

Candidate No: 105936

12th January 2019

Contents

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 focused on 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. Please note a digital copy of this document can be found at `https://github.com/jamesfernando94/UniversityDocuments/raw/master/4-RealTimeEmbeddedSystems/FinalReport/RealTimeEmbeddedSystemsFinalReport.pdf` it is advised to use this due to the number of links in the document.

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 ensuring that there is no delay for the program in terms of using interrupts correctly and ensuring 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 to complete the project.

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 complied to run on the specific hardware. This is sightly different from my background in programming for computers where a program is complied 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, I found the interrupts to be very useful and vital 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 fo 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 exercises although it did allow for the learning of a number of parts of serial communications, such as the purpose of different lines. 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 ideas for what I could produce for my project. The IBM application clearly showed what sensors were 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 via it's IP address and then query a external API for the current weather conditions before displaying the information on the screen. The code for the application can be found at section C and the

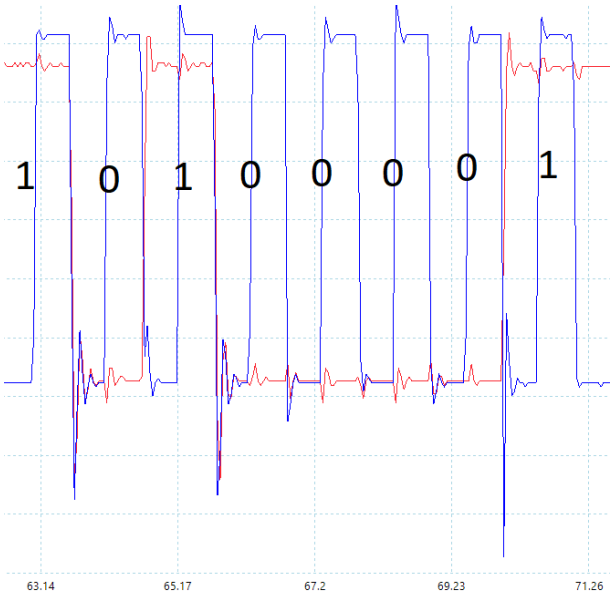

Figure 1: An Image showing the output on the oscilloscope when outputting the byte 10100001

full repository can be accessed at `https://os.mbed.com/users/jamesfernando/code/frdm_project/`.

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]. This 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. The code I used for this task can be found at section B.1.

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. The code I used for this can be found at section B.2.

4.3. Re-factoring the code

After implementing the function to get the IP address and display it to the screen I realised that if I wanted to display each of the individual pieces of information I would have to change the layout of the code to make this easier. 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. The code I used for this can be found at section B.3.

4.5. Getting the Location

Obtaining the location was simple as 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 to 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. The code I used for this can be found at section B.4.

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]. The code I used for this can be found at section B.5.

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 times 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 default 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

```
/* Program: A simple Timer Activate Tera Term terminal to test.*/
#include "mbed.h"
#include <string>

Timer t; // define Timer with name "t"
Serial pc(USBTX, USBRX);
int main()
{
    for(int i = 0; i <= 10; i++) { // Test 1 to 10 characters
        string str = "";
        for(int j = 0; j < i; j++) { // Add i characters to string str
            str += "a";
        }
        t.reset();
        t.start(); //start the timer
        pc.printf("%s\n", str.c_str());
        t.stop(); //stop the timer
        pc.printf("The time taken for %i characters was %f seconds\n", i, t.read()); //print to pc
    }
}
```

B. Program Extracts

B.1. Get IP address via HTTP code

```
/*
*   This function gets the current IP Address and stores it in the ipAddress variable
*/
void set_ip_address(NetworkInterface *network)
{
    HttpRequest *request = new HttpRequest(network, HTTP_GET, "http://api.ipify.org/"); // Setup http Request
    request->set_header("Content-Type", "application/json");
    HttpResponse *response = request->send(); // send off request and store response in response vairable
    dump_response(response);
    ipAddress = response->get_body_as_string(); // get the body of the response which contains the IP Address
    delete request; // also clears out the response
    display_data();
}
```

B.2. Display to LCD

```
/**
*   This function displays the correct data depending on the position variable.
```

```
*/
void display_data(){
    switch (position){
        case 0:
            display_ip_address();
            break;
        case 1:
            display_location();
            break;
        case 2:
            display_temprature();
            break;
        case 3:
            display_weather();
        default:
            break;
    }
}

/*
*   This function displays the IP Address
*/
void display_ip_address()
{
    lcd.cls();
    lcd.locate(0, 3);
    lcd.printf("IP Address: %s", ipAddress.c_str());
}

/*
*   This function displays the Latitude and logitude
*/
void display_location()
{
    lcd.cls();
    lcd.locate(0, 3);
    lcd.printf("Latitude: %s\n", latitude.c_str());
    lcd.printf("Longitude: %s", longitude.c_str());
}

/*
*   This function displays the temprature
*/
void display_temprature()
{
    lcd.cls();
    lcd.locate(0, 3);
    lcd.printf("Current Temprature: %s", temprature.c_str());
}

/*
* This function displays the wether description
*/
void display_weather(){
    lcd.cls();
    lcd.locate(0, 3);
    lcd.printf("Weather Description: %s", weatherDescription.c_str());
}
```

B.3. Using the Joystick

```
int main()
{
    Thread eventThread; // Set up thread to add events to
    eventThread.start(callback(&queue, &EventQueue::dispatch_forever)); //
    add queue to thread

    display_data();
    joy_up.rise(queue.event(&up)); // This adds the up function to the
    queue when the interrupt is called
    joy_down.rise(queue.event(&down)); // This adds the down function to
    the queue when the interrupt is called
    // Connect to the network with the default networking interface
    // if you use WiFi: see mbed_app.json for the credentials
    NetworkInterface *network = connect_to_default_network_interface();
    if (!network)
    {
        printf("Cannot connect to the network, see serial output\n");
        return 1;
    }

    // Set data values
    set_ip_address(network);
    set_location(network);
    set_weather(network);

    // Wait forever note as the interrupts work on a different thread this
    is fine
    wait(osWaitForever);
}

/*
*   This function is the up funciton it decrements the position value or
    goes round if zero
*/
void up()
{
    printf("\n Called Up");
    if (position <= 0)
    {
        position = MAX_POSITIONS - 1;
    }
    else
    {
        position--;
    }
    display_data();
}

/*
*   This function increments the position value up to MAX_POSITIONS
*/
void down()
{
    printf("\n Called Down");
    position++;
    position %= MAX_POSITIONS;
```

```
52      display_data();
53  }
```

B.4. Getting the Location

```
1   /*
2   *   This function gets the location based on the IP Address and stores it
        in the latitude and logitude variables
3   */
4   void set_location(NetworkInterface *network){
5       MbedJSONValue location;
6       string url = "http://api.ip2location.com/?ip=" + ipAddress + "&key=
        UMJGR0FYSZ&package=WS5&format=json"; // build request url
7       HttpRequest *request = new HttpRequest(network, HTTP_GET, url.c_str());
        // Setup http Request
8       request->set_header("Content-Type", "application/json");
9       HttpResponse *response = request->send(); // send off request and store
         response in response vairable
10      dump_response(response);
11      parse(location, response->get_body_as_string().c_str()); // convert
        body from JSON string to Json object
12      latitude = location["latitude"].get<string>(); // Get latitude from
        JSON object
13      longitude = location["longitude"].get<string>(); // Get longitude from
        JSON object
14      delete request; // also clears out the response
15      display_data();
16  }
```

B.5. Getting the Weather Information

```
1   /*
2   *   This function gets the weather information based on the location
        information and stores it in the global variables
3   */
4   void set_weather(NetworkInterface *network){
5       MbedJSONValue weather;
6       string url = "http://api.openweathermap.org/data/2.5/weather?lat=" +
        latitude + "&lon=" + longitude + "&appid=8218
        d0e6eee9e8b53f74a4a646145ea5&units=metric"; // build request url
7       HttpRequest *request = new HttpRequest(network, HTTP_GET, url.c_str());
         // Setup http Request
8       request->set_header("Content-Type", "application/json");
9       HttpResponse *response = request->send(); // send off request and store
         response in response vairable
10      dump_response(response);
11      parse(weather, response->get_body_as_string().c_str()); // convert body
         from JSON string to Json object
12      double temp = weather["main"]["temp"].get<double>(); // Get Temprature
        as doble and store it in tempoary variable
13      // Convert double to string and append " C" to the end
14      std::ostringstream strs;
15      strs << temp;
16      strs << " C";
17      temprature = strs.str();
18
19      weatherDescription = weather["weather"][0]["description"].get<string>()
        ; // Store Weather description in variable
```

```
    delete request; // also clears out the response
    display_data();
}
```

C. Weather App Project

Please note that the full repository can be found at `https://os.mbed.com/users/jamesfernando/code/frdm_project/`

```

#include "mbed.h"
#include "http_request.h"
#include "network-helper.h"
#include "mbed_mem_trace.h"
#include <string>
#include "C12832.h"
#include "MbedJSONValue.h"
#include <sstream>

// LCD
C12832 lcd(D11, D13, D12, D7, D10);
// Joystick Inputs
InterruptIn joy_up(A2);
InterruptIn joy_down(A3);

// Global Variables with inital values
int position = 0;
string ipAddress = "waiting...";
string latitude = "waiting...";
string longitude = "waiting...";
string temprature = "waiting...";
string weatherDescription = "waiting...";
EventQueue queue;

// Constants
const int MAX_POSITIONS = 4;

// Method Declarations
void display_data();
void up();
void down();
void display_ip_address();
void display_location();
void display_temprature();
void display_weather();
void set_ip_address(NetworkInterface *network);
void set_location(NetworkInterface *network);
void set_weather(NetworkInterface *network);
void dump_response(HttpResponse *res);

int main()
{
    Thread eventThread; // Set up thread to add events to
    eventThread.start(callback(&queue, &EventQueue::dispatch_forever)); //
    add queue to thread

    display_data();
```

```
48      joy_up.rise(queue.event(&up)); // This adds the up function to the
        queue when the interrupt is called
49      joy_down.rise(queue.event(&down)); // This adds the down function to
        the queue when the interrupt is called
50      // Connect to the network with the default networking interface
51      // if you use WiFi: see mbed_app.json for the credentials
52      NetworkInterface *network = connect_to_default_network_interface();
53      if (!network)
54      {
55          printf("Cannot connect to the network, see serial output\n");
56          return 1;
57      }
58
59      // Set data values
60      set_ip_address(network);
61      set_location(network);
62      set_weather(network);
63
64      // Wait forever note as the interrupts work on a different thread this
        is fine
65      wait(osWaitForever);
66  }
67
68  /**
69  *   This function displays the correct data depending on the position
        variable.
70  */
71  void display_data(){
72      switch (position){
73          case 0:
74              display_ip_address();
75              break;
76          case 1:
77              display_location();
78              break;
79          case 2:
80              display_temprature();
81              break;
82          case 3:
83              display_weather();
84          default:
85              break;
86      }
87  }
88
89  /*
90  *   This function is the up funciton it decrements the position value or
        goes round if zero
91  */
92  void up()
93  {
94      printf("\n Called Up");
95      if (position <= 0)
96      {
97          position = MAX_POSITIONS - 1;
98      }
99      else
100     {
```

```
101             position--;
102         }
103         display_data();
104 }
105 
106 /*
107 *   This function increments the position value up to MAX_POSITIONS
108 */
109 void down()
110 {
111         printf("\n Called Down");
112         position++;
113         position %= MAX_POSITIONS;
114         display_data();
115 }
116 
117 /*
118 *   This function displays the IP Address
119 */
120 void display_ip_address()
121 {
122         lcd.cls();
123         lcd.locate(0, 3);
124         lcd.printf("IP Address: %s", ipAddress.c_str());
125 }
126 
127 /*
128 *   This function displays the Latitude and logitude
129 */
130 void display_location()
131 {
132         lcd.cls();
133         lcd.locate(0, 3);
134         lcd.printf("Latitude: %s\n", latitude.c_str());
135         lcd.printf("Longitude: %s", longitude.c_str());
136 }
137 
138 /*
139 *   This function displays the temprature
140 */
141 void display_temprature()
142 {
143         lcd.cls();
144         lcd.locate(0, 3);
145         lcd.printf("Current Temprature: %s", temprature.c_str());
146 }
147 
148 /*
149 * This function displays the wether description
150 */
151 void display_weather(){
152         lcd.cls();
153         lcd.locate(0, 3);
154         lcd.printf("Weather Description: %s", weatherDescription.c_str());
155 }
156 
157 /*
```

```
158 *   This function gets the current IP Address and stores it in the
    ipAddress variable
159 */
160 void set_ip_address(NetworkInterface *network)
161 {
162     HttpRequest *request = new HttpRequest(network, HTTP_GET, "http://api.
    ipify.org/"); // Setup http Request
163     request->set_header("Content-Type", "application/json");
164     HttpResponse *response = request->send(); // send off request and store
     response in response vairable
165     dump_response(response);
166     ipAddress = response->get_body_as_string(); // get the body of the
    response which contains the IP Address
167     delete request; // also clears out the response
168     display_data();
169 }
170
171 /*
172 *   This function gets the location based on the IP Address and stores it
    in the latitude and logitude variables
173 */
174 void set_location(NetworkInterface *network){
175     MbedJSONValue location;
176     string url = "http://api.ip2location.com/?ip=" + ipAddress + "&key=
    UMJGR0FYSZ&package=WS5&format=json"; // build request url
177     HttpRequest *request = new HttpRequest(network, HTTP_GET, url.c_str());
    // Setup http Request
178     request->set_header("Content-Type", "application/json");
179     HttpResponse *response = request->send(); // send off request and store
     response in response vairable
180     dump_response(response);
181     parse(location, response->get_body_as_string().c_str()); // convert
    body from JSON string to Json object
182     latitude = location["latitude"].get<string>(); // Get latitude from
    JSON object
183     longitude = location["longitude"].get<string>(); // Get longitude from
    JSON object
184     delete request; // also clears out the response
185     display_data();
186 }
187
188 /*
189 *   This function gets the weather information based on the location
    information and stores it in the global variables
190 */
191 void set_weather(NetworkInterface *network){
192     MbedJSONValue weather;
193     string url = "http://api.openweathermap.org/data/2.5/weather?lat=" +
    latitude + "&lon=" + longitude + "&appid=8218
    d0e6eee9e8b53f74a4a646145ea5&units=metric"; // build request url
194     HttpRequest *request = new HttpRequest(network, HTTP_GET, url.c_str());
     // Setup http Request
195     request->set_header("Content-Type", "application/json");
196     HttpResponse *response = request->send(); // send off request and store
     response in response vairable
197     dump_response(response);
198     parse(weather, response->get_body_as_string().c_str()); // convert body
     from JSON string to Json object
```

```
199     double temp = weather["main"]["temp"].get<double>(); // Get Temprature
        as doble and store it in tempoary variable
200     // Convert double to string and append " C" to the end
201     std::ostringstream strs;
202     strs << temp;
203     strs << " C";
204     temprature = strs.str();
205
206     weatherDescription = weather["weather"][0]["description"].get<string>()
        ; // Store Weather description in variable
207     delete request; // also clears out the response
208     display_data();
209 }
210
211 /*
212 *   This function dumps the HTTPResponse object to the console connected to
        the debug port
213 */
214 void dump_response(HttpResponse *res)
215 {
216     printf("Status: %d - %s\n", res->get_status_code(), res->
        get_status_message().c_str());
217
218     printf("Headers:\n");
219     for (size_t ix = 0; ix < res->get_headers_length(); ix++)
220     {
221         printf("\t%s: %s\n", res->get_headers_fields()[ix]->c_str(), res->
        get_headers_values()[ix]->c_str());
222     }
223     printf("\nBody (%d bytes):\n\n%s\n", res->get_body_length(), res->
        get_body_as_string().c_str());
224 }
```

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).