Sheffield Hallam University
Department of Engineering
BEng (Hons) Electrical and Electronic Engineering

Activity ID		Activity Title			Laboratory Room No.	Level
Lab 104		Dealing with data			4302	6
Semester	**Duration [hrs]**	**Group Size**	**Max Total Students**	**Date of approval**	**Lead Academic**	
1	6	1	25	09-22	Alex Shenfield	

Equipment (per student/group)

Number	Item
1	PC
1	Arduino kit

Learning Outcomes

	Learning Outcome
2	Demonstrate an understanding of the various tools, technologies and protocols used in the development of embedded systems with network functionality
3	Design, implement and test embedded networked devices

Dealing with Data using the Arduino

Introduction

Computing is about more than the PC on your desktop! Embedded devices are everywhere – from wireless telecommunications infrastructure points to electronic point of sale terminals. One definition of an embedded system is:

> "**An embedded system is a computer system designed to perform one or a few dedicated functions** often with real-time computing constraints."
> (http://en.wikipedia.org/wiki/Embedded_system)

In this series of labs you are going to be introduced to the open source Arduino platform – a cheap, simple to program, well documented prototyping platform for designing electronic systems. Although the Arduino platform started out with the Arduino Uno (based on the commonly used ATMega328 chip), other, more capable, boards have since been released. In this series of labs you will be using the Arduino MKR WiFi 1010 board – a Cortex M0+ based Arduino board with built in WiFi module.

So far in the laboratory sessions we have provided a gentle introduction to both the software and hardware aspects of the Arduino platform, looked at some of types of sensors we can use with Arduino, and introduced the mechanism of producing an analog output from a digital microcontroller using PWM.

In this laboratory session we are going to investigate ways of dealing with data using the Arduino MKR WiFi 1010 – including **displaying it** and **logging it**.

By the end of this set of laboratory exercises you will be able to use both the SPI and I2C protocols to interact with the features on the Adafruit 2.8" colour TFT + capacitive touchscreen breakout board[1]. This board allows us to create attractive on-device user interfaces to provide integral display of data, a touch screen to allow us to interact with UI elements such as buttons, and an SD card interface to allow us to log data points with time.

Bibliography

There are no essential bibliographic resources for this laboratory session aside from this tutorial sheet. However the following websites and tutorials may be of help (especially if you haven't done much electronics previously or your digital logic and/or programming is a bit rusty):

- http://www.arduino.cc/
- http://www.ladyada.net/learn/arduino/index.html
- http://tronixstuff.wordpress.com/tutorials/

1 Product page here: https://www.adafruit.com/product/2090

Methodology

Check that you have all the necessary equipment (see Figure 1)!

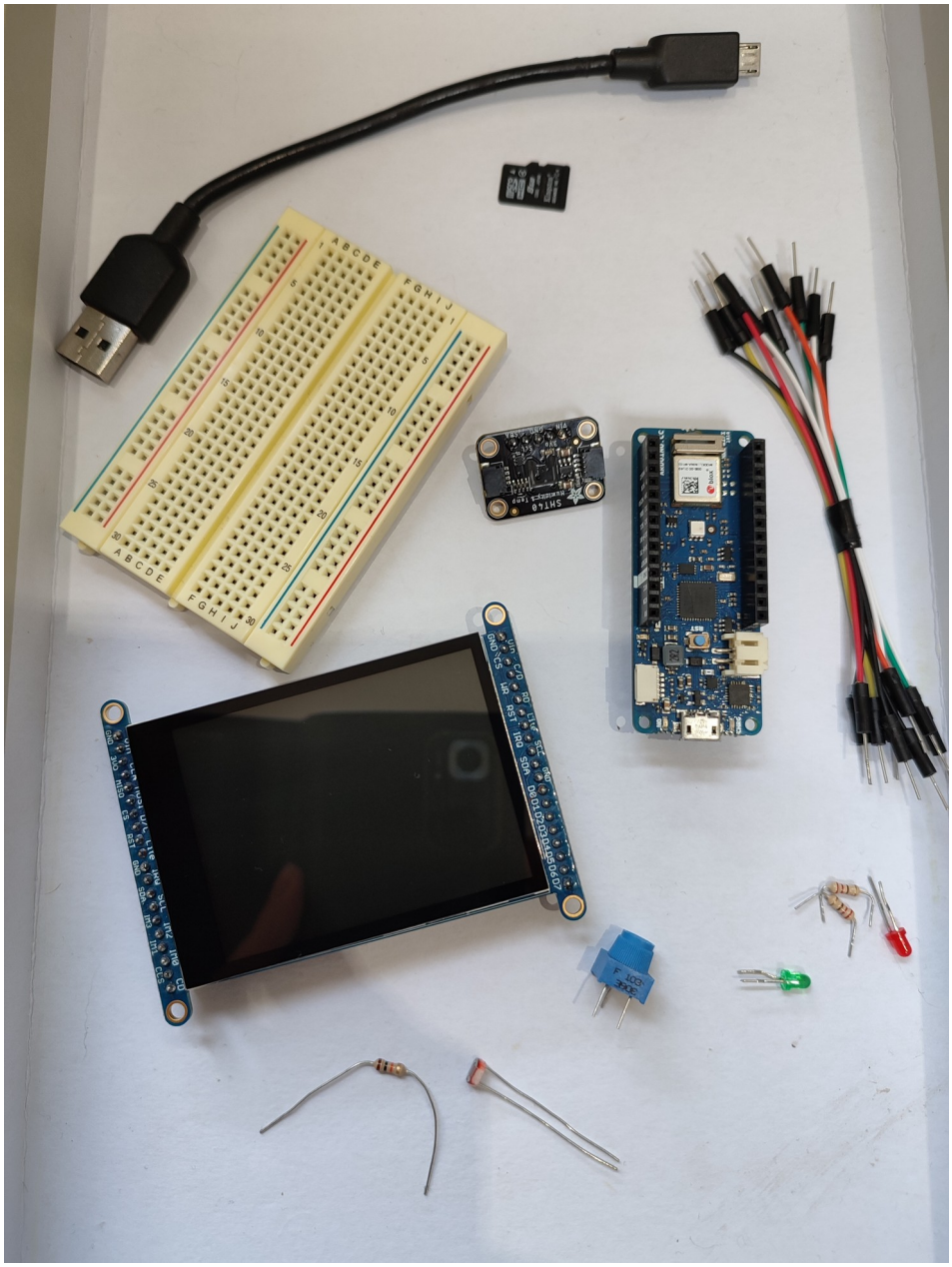

Figure 1 – The necessary equipment for this lab

Task 1

The Arduino development kits provided for this module have a 2.8” TFT LCD breakout board with capacitive touch screen included with them. This breakout board has a controller built into it that uses RAM buffering – meaning that almost no work is done by the microcontroller. The display can be used in two modes: 8-bit and SPI. In these labs we will focus on using the SPI mode as it requires only 5 pins in total (SPI data in, data out, clock, select, and a data / command pin).

This LCD screen allows us to draw shapes, display data as text, and even to render bitmaps! To make our life easy we will use several libraries to interact with this display:

1. The Adafruit ILI9341 TFT Library – which provides low level code specific to the TFT controller chip built-in to this device.
2. The Adafruit GFX Library[2] – which provides a whole load of convenience functions for manipulating pixels, drawing shapes, printing text, and even loading images.

Install these libraries using the Arduino library manager (as we’ve done in previous labs). This will also include any dependencies needed to use those libraries.

Now connect this breakout board to the Arduino MKR WiFi 1010 as shown in Figure 2.

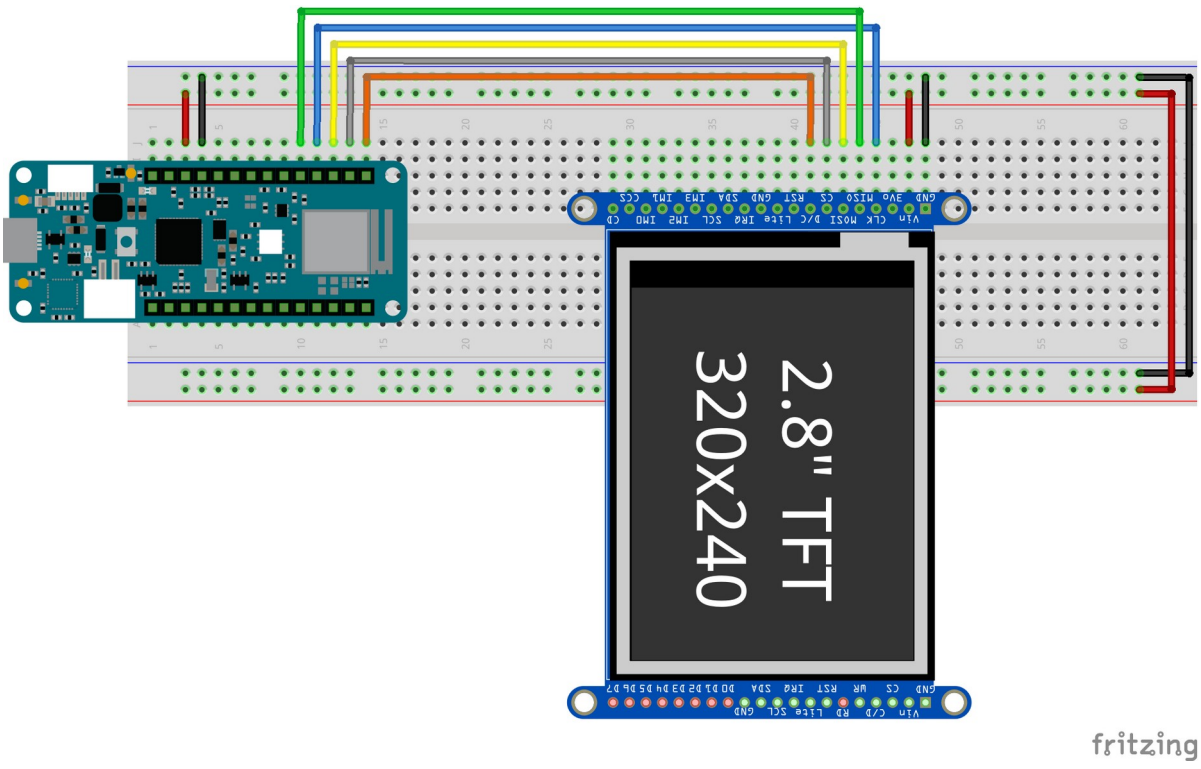

Figure 2 – Arduino MKR WiFi 1010 connected to the Adafruit 2.8" TFT LCD with capacitive touch screen

2 A full description of all the cool features supplied by the Adafruit GFX library is available here: https://learn.adafruit.com/adafruit-gfx-graphics-library/overview

Figure 3 shows the LCD display showing a simple “hello world” message and uptime counter (code for this application is provided in code listing 1). This uptime counter is updated every second.

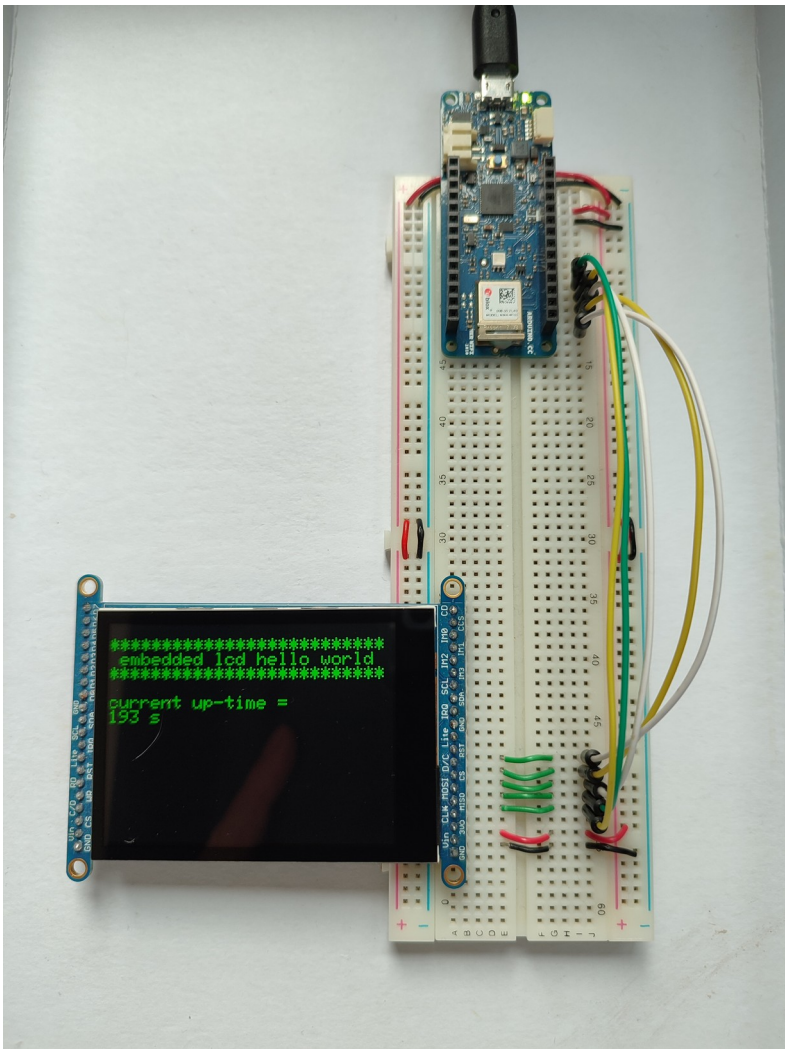

Figure 3 – The LCD “hello world” program in action

Code listing 1: Code for the simple lcd hello world program

```
/**
* 1_lcd_hello_world.ino
*
* this is a simple Arduino program to keep track of the up-time using
* and display it on the Adafruit 2.8" TFT LCD screen:
*
* https://www.adafruit.com/product/2090
*
* author: alex shenfield
* date:   08-10-2022
*/

// LIBRARY IMPORTS

// include the necessary libraries for the LCD
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_ILI9341.h>

// INITIAL VARIABLES

// DISPLAY

// set the additional SPI pins for the TFT break out
#define TFT_DC 6
#define TFT_CS 7

// create the tft object using the hardware SPI pins
Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC);

// UPTIME COUNTER

// timing variables ...
long previous_time = 0;
long interval = 1000;

// create a variable to increment every second
int current_uptime = 0;
```

```
// CODE

// this method runs once (when the sketch starts)
void setup()
{
  // start serial comms and wait until its up and running
  Serial.begin(115200);
  while (!Serial)
  {
    delay(10);
  }
  Serial.println("LCD hello world program running ...");

  // start the tft screen and display some diagnostic data on the
  // serial display
  tft.begin();
  print_diagnostics();

  // make the screen black
  tft.fillScreen(ILI9341_BLACK);

  // set the text colour, background colour, and text size
  tft.setTextColor(ILI9341_GREEN, ILI9341_BLACK);
  tft.setTextSize(2);

  // rotate the display, reset the cursor to the top left, and write
  // a header to it
  tft.setRotation(1);
  tft.setCursor(0, 0);
  tft.println();
  tft.println("**************************");
  tft.println(" embedded lcd hello world ");
  tft.println("**************************");
  tft.println();
}

// this method loops continuously
void loop()
{
  // get the current time this time round the loop
  unsigned long current_time = millis();

  // if the set time interval has elapsed ...
  if (current_time - previous_time > interval)
  {
    // save the time
    previous_time = current_time;

    // update the current uptime
    current_uptime = current_uptime + 1;

    // set the cursor to the start of the fifth line - each character
    // is 16 pixels high and so the cursor should be set to y = 5 * 16
    tft.setCursor(0, 80);
    tft.println("current up-time = ");
    tft.print(current_uptime);
    tft.println(" s");
  }
}
```

```
// UTILITY METHODS

// print some simple diagnostic information about the TFT LCD
// (this is optional but can help debug problems - e.g. the lcd not
// properly connected)
void print_diagnostics()
{
  uint8_t x = 0;
  x = tft.readcommand8(ILI9341_RDMODE);
  Serial.print("display power mode: 0x");
  Serial.println(x, HEX);
  x = tft.readcommand8(ILI9341_RDMADCTL);
  Serial.print("MADCTL mode: 0x");
  Serial.println(x, HEX);
  x = tft.readcommand8(ILI9341_RDPIXFMT);
  Serial.print("pixel format: 0x");
  Serial.println(x, HEX);
  x = tft.readcommand8(ILI9341_RDIMGFMT);
  Serial.print("image format: 0x");
  Serial.println(x, HEX);
  x = tft.readcommand8(ILI9341_RDSELFDIAG);
  Serial.print("self diagnostic: 0x");
  Serial.println(x, HEX);
}
```

Enter this program, upload it to the board, and make sure everything works.

Figure 4 shows the Serial Monitor output for this program – note the debugging information displayed. Sometimes the wiring for the LCD can be a bit finicky and, if that happens, the debugging information will not be as complete (you'll probably see "0x0" in some places).

Figure 4 – Serial Monitor output showing LCD debugging information.

Exercises

Now you are going to make some modifications to the program to add functionality:

a) Add a potentiometer into the system and display the value on the LCD screen.

b) Try and remap the potentiometer values to be within certain bounds (e.g. between 0 and 100). Use these ideas to make the LCD screen show a percentage for the potentiometer value.

c) Now add some LEDs to your circuit and use the potentiometer to control when the LEDs come on. Output the status of the LEDs to the LCD screen.

d) Try outputting the potentiometer values to the LCD screen as a bar graph. You should look to scaling the potentiometer values and using them to create a rectangle[3]. I would **highly** recommend using pseudocode or other algorithm design methodologies before actually writing the code! Figure 5 shows an example of what this might look like.

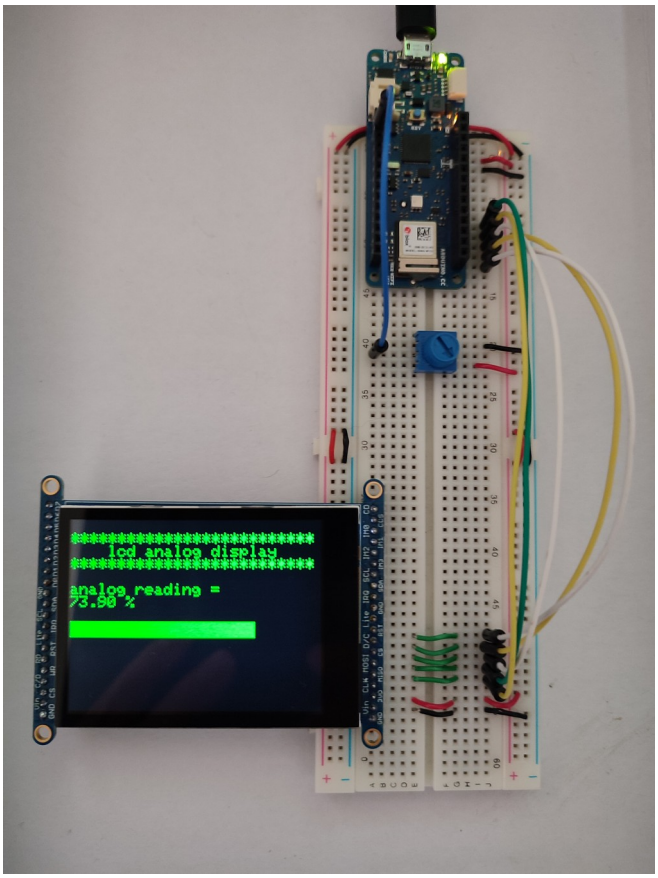

Figure 5 – An example bar graph display on the Arduino MKR WiFi 1010 and Adafruit 2.8" TFT display

3 The *Adafruit_GFX* library provides a:

```
void fillRect(uint16_t x0, uint16_t y0, uint16_t w, uint16_t h, uint16_t color);
```

function that allows you to draw a rectangle of a specified width and height. In this function x0is the horizontal alignment and y0 is the vertical alignment.

Task 2

As well as a 240 x 320 pixel graphical TFT LCD screen, the breakout board is also touch sensitive (using a capacitive touch screen). This allows many different potential uses including virtual key pads and other controls.

To use this touch screen capability, we need to wire up the I2C pins on the breakout board to the I2C pins on the Arduino MKR WiFi 1010 – make the additional connections to this breakout board as shown in Figure 6 (see highlighted area in red).

We will also have to install the FT6206 controller library which does all the low level communication with the FT6206 driver chip.

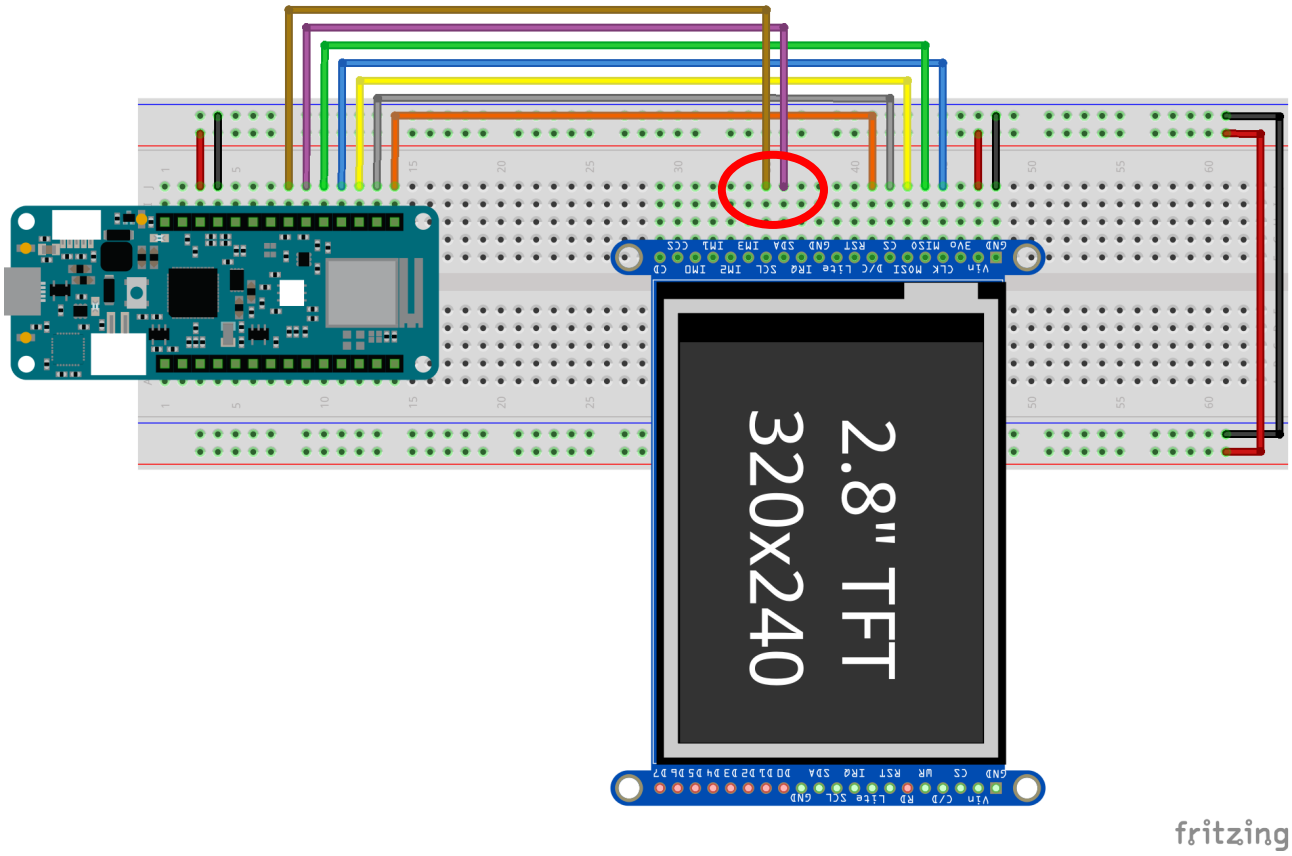

Figure 6 – Fritzing breadboard schematic of the TFT LCD breakout board with the touch screen enabled

In this simple example, we will implement a touch sensitive button and a virtual led that changes colour when the button is pressed (as shown in Figure 7).

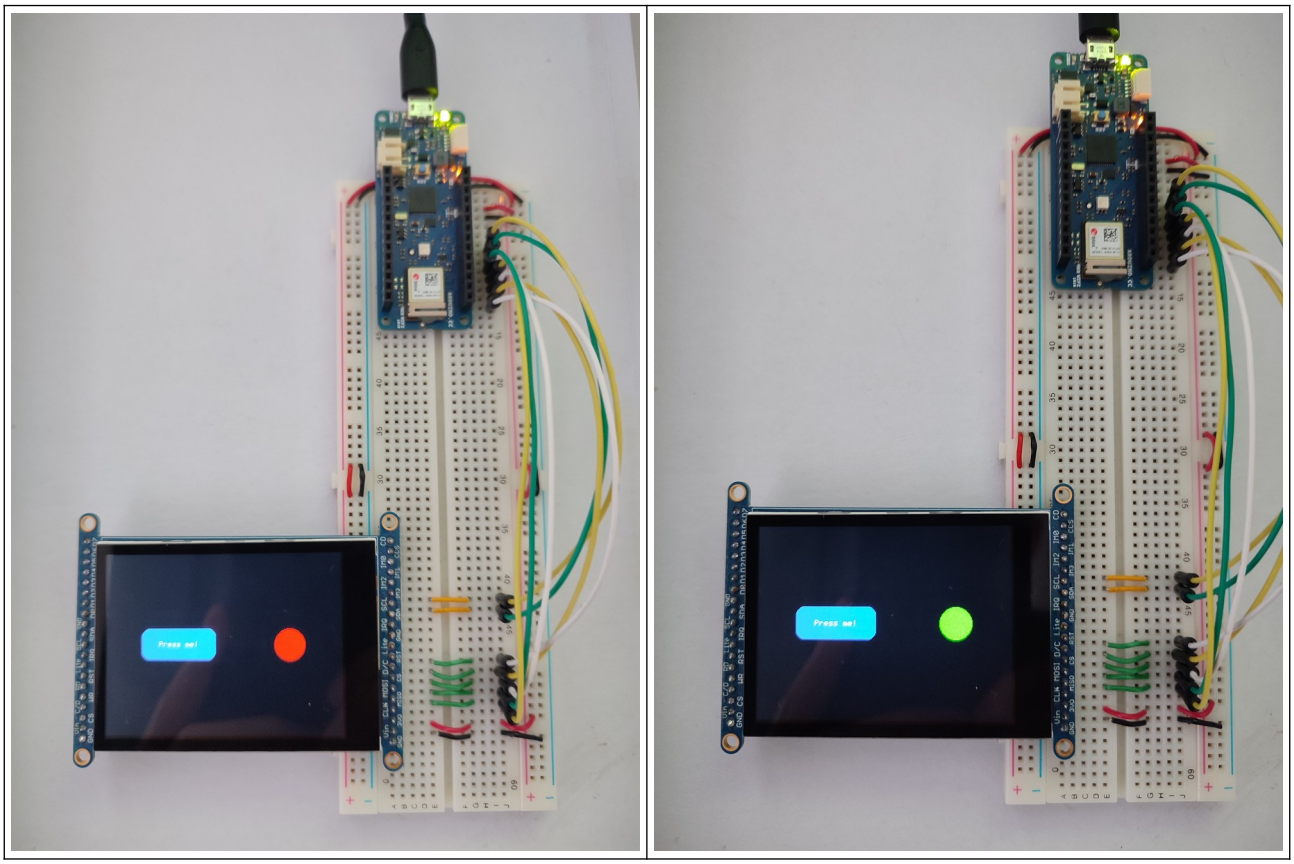

Figure 7 – The touchscreen example

Code listing 2 provides the code for this “virtual led” example.

Code listing 2: Code for the touch screen virtual led program

```
/**
* 2_tft_touchscreen_example.ino
*
* a simple touch screen demo using the adafruit gfx libraries and
* drivers for the adafruit 2.8" TFT + capactive touchscreen display
* breakout:
*
* https://www.adafruit.com/product/2090
*
* this demo has a touchable button that controls a virtual led. touch
* the button and the led turns on, touch it again and it turns off.
*
* full documentation for all features can be found at:
*
* https://adafruit.github.io/Adafruit-GFX-Library/html/index.html
*
* author: alex shenfield
* date:   12/10/2022
*/

// include the necessary libraries
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_ILI9341.h>
#include <Adafruit_FT6206.h>

// INITIAL DEFINES

// LCD AND TOUCHSCREEN

// define the custom pins
#define TFT_DC 6
#define TFT_CS 7

// create display and touchscreen objects
Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC);
Adafruit_FT6206   ts = Adafruit_FT6206();

// lcd size
const int lcd_width  = 320;
const int lcd_height = 240;
```

```
// GRAPHICS OBJECTS

// define the button
Adafruit_GFX_Button on_button;

// set the x and y positions to put the button on the centre left of
// the screen (note: the button coordinates are the centre of the
// button)
const int button_x = 100;
const int button_y = 120;

// set the height and width of the button
const int button_h = 40;
const int button_w = 100;

// set the x and y positions so that the "led" is centre right of the
// screen (note: the coordinates are the centre of the circle)
const int led_x = 250;
const int led_y = 120;

// led radius
const int led_r = 20;

// STATE VARIABLES

// keep track of the led state
int led_state = LOW;

// CODE

// this method runs once (when the sketch starts)
void setup()
{
  // start the touchscreen and the tft screen and make the screen black
  ts.begin();
  tft.begin();
  tft.setRotation(1);
  tft.fillScreen(ILI9341_BLACK);

  // draw the button
  on_button.initButton(&tft, button_x, button_y, button_w, button_h,
                       ILI9341_BLUE, ILI9341_WHITE, ILI9341_BLUE,
                       "Press me!", 1, 1);
  on_button.drawButton(true);

  // draw the "led"
  tft.fillCircle(led_x, led_y, led_r, ILI9341_RED);
}
```

```
// this method loops continuously
void loop()
{
  // if the touch screen has been pressed ...
  if (ts.touched())
  {
    // get the touch coordinate and flip it to match the orientation
    // of the screen
    TS_Point p = ts.getPoint();
    p.x = map(p.x, 0, lcd_height, lcd_height, 0);
    p.y = map(p.y, 0, lcd_width, lcd_width, 0);
    int y = tft.height() - p.x;
    int x = p.y;

    // check the button
    if (on_button.contains(x,y))
    {
      on_button.press(true);
    }
  }
  // as soon as it's released, reset the button press
  else
  {
    on_button.press(false);
  }

  // if the button was pressed since we last checked the touch screen
  // state then toggle the led colour
  if (on_button.justPressed())
  {
    if (led_state == LOW)
    {
      tft.fillCircle(led_x, led_y, led_r, ILI9341_GREEN);
      led_state = HIGH;
    }
    else
    {
      tft.fillCircle(led_x, led_y, led_r, ILI9341_RED);
      led_state = LOW;
    }
  }
}
```

Enter this program, upload it to the board, and make sure everything works.

Exercises

Now you are going to make some modifications to the program to add functionality:

a) Add an additional button and use it to increment a counter variable which is displayed on the screen.

b) When this counter variable exceeds a certain value, light up a virtual LED.

c) Wire up a real LED and light this up too.

Figure 8 (below) shows an example of what this completed system may look like.

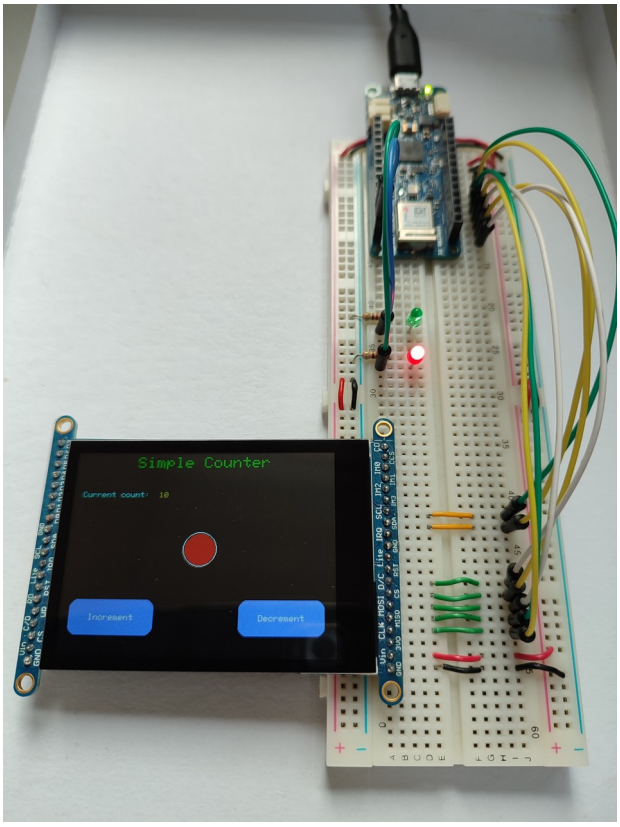

Figure 8 – Touchscreen counter example

Task 3

An important part of logging data is the ability to correlate the data points with timestamps. This allows you to see time varying trends in your data – for example, how temperatures vary throughout the day. Whilst the Arduino platform does have the **`millis()`** function that allows us to keep track of short time periods, the Arduino MKR WiFi 1010 board also has a built in real-time clock that allows us to keep accurate track of the current time over long time periods – even when the board is in low-power sleep mode[4].

To simplify the process of interacting with this built-in RTC, I have written my own library on top of the standard RTCZero library. My library provides some simple methods for automatically setting the time and date (rather than having to manually specify the day / month / year / hour / minute / seconds every time you compile the program) and also some neat methods for returning nicely formatted date and time strings for printing out (e.g. to the Serial Monitor).

Firstly you will have to install the RTCZero library using the Arduino library manager. We will then create additional tabs for the library files (one for the header file, and one for the actual library code). Figures 9 - 11 show this process.

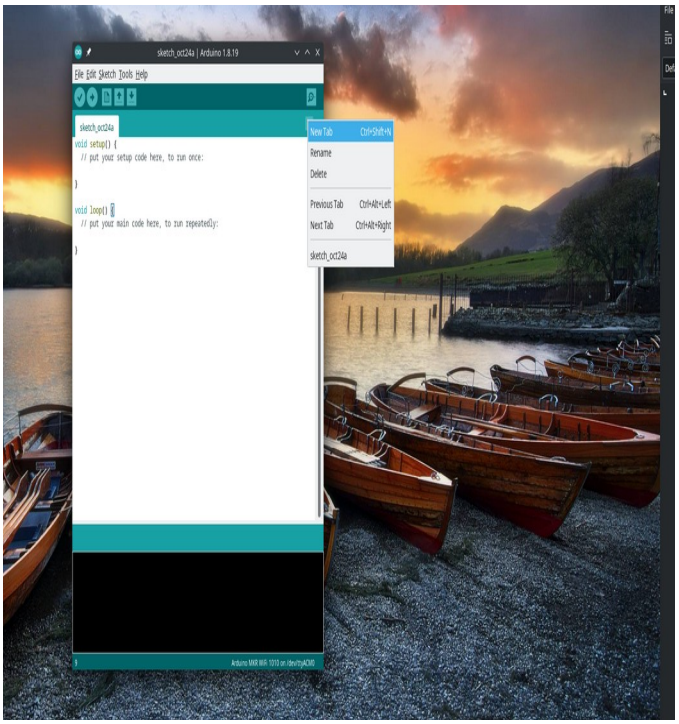

Figure 9 – Creating a new file in the Arduino IDE

4 To keep the RTC running it is necessary to keep the board powered. However, we can use a button sized lithium battery (or any battery in the 3V range) connected through a diode to the 3.3V pin to keep RTC alive if the board is put in sleep mode before the standard USB or VIN power is disconnected.

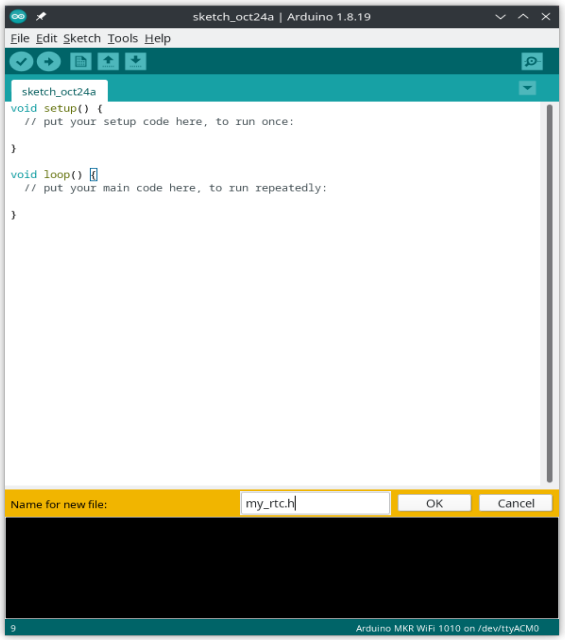

Figure 10 – Naming the new file in the Arduino IDE

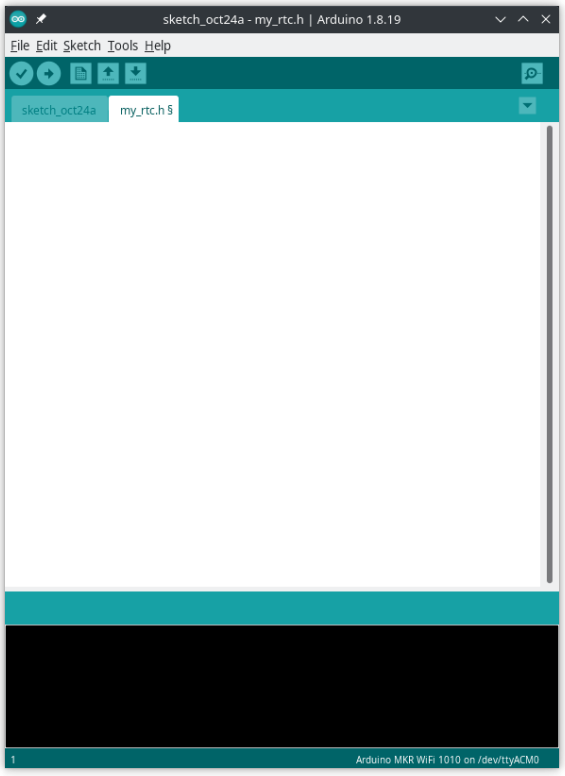

Figure 11 – A new file in the Arduino IDE (in a separate tab)

Finally, Figure 12 shows the completed IDE layout.

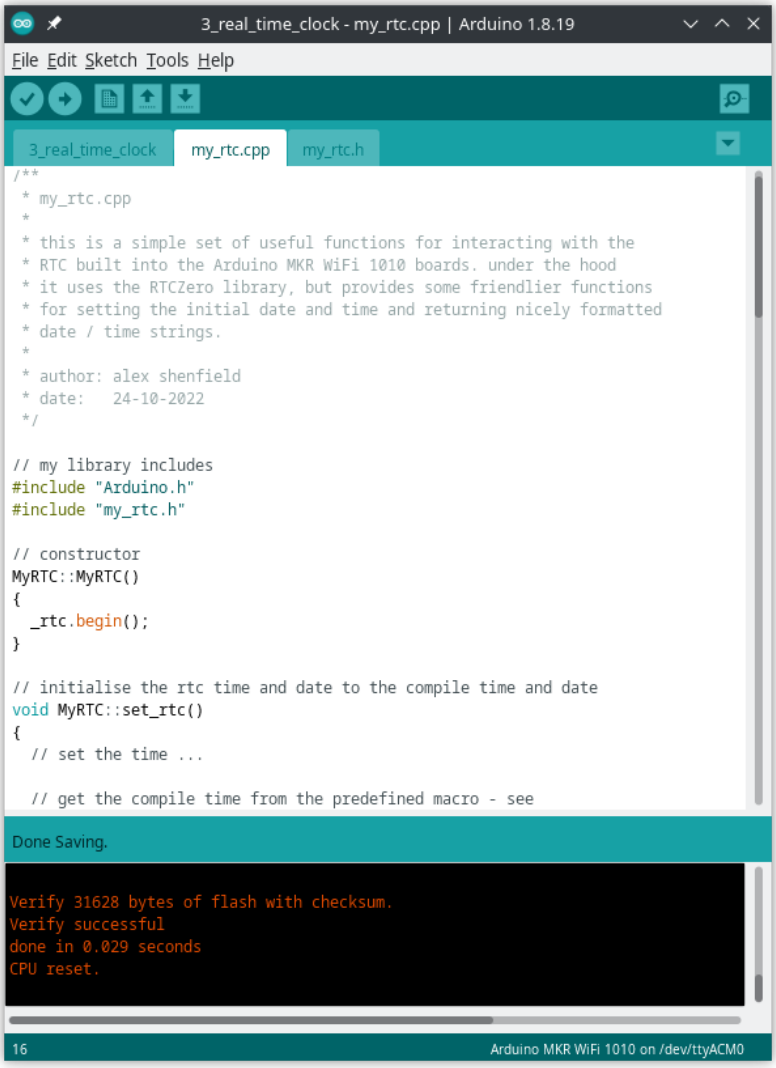

Figure 12 – The RTC project with my_rtc library

Now we need to write the code for the new RTC library. The code for my_rtc.h (the header file) is shown in code listing 3 and the code for the actual library functions is shown in code listing 4.

Code listing 3: Code for my_rtc.h

```
/**
* my_rtc.h
*
* this is a simple set of useful functions for interacting with the
* RTC built into the Arduino MKR WiFi 1010 boards. under the hood
* it uses the RTCZero library, but provides some friendlier functions
* for setting the initial date and time and returning nicely formatted
* date / time strings.
*
* author: alex shenfield
* date:   24-10-2022
*/

// prevent recursive includes
#ifndef MyRTC_h
#define MyRTC_h

// include the basic Arduino commands and the RTCZero library for access
// to the MKR WiFi 1010 RTC
#include "Arduino.h"
#include <RTCZero.h>

// this is my rtc library
class MyRTC
{
public:
    MyRTC();
    void set_rtc();
    String get_rtc_time();
    String get_rtc_date();
private:
    RTCZero _rtc;
};

#endif
```

Code listing 4: Code for my_rtc.cpp

```
/**
* my_rtc.cpp
*
* this is a simple set of useful functions for interacting with the
* RTC built into the Arduino MKR WiFi 1010 boards. under the hood
* it uses the RTCZero library, but provides some friendlier functions
* for setting the initial date and time and returning nicely formatted
* date / time strings.
*
* author: alex shenfield
* date:   24-10-2022
*/

// my library includes
#include "Arduino.h"
#include "my_rtc.h"

// constructor
MyRTC::MyRTC()
{
  _rtc.begin();
}

// initialise the rtc time and date to the compile time and date
void MyRTC::set_rtc()
{
  // set the time ...

  // get the compile time from the predefined macro - see
  // https://gcc.gnu.org/onlinedocs/cpp/Standard-Predefined-Macros.html
  // e.g. 10:15:30
  String t = __TIME__;
  uint8_t hours   = t.substring(0, 2).toInt();
  uint8_t minutes = t.substring(3, 5).toInt();
  uint8_t seconds = t.substring(6, 8).toInt();
  _rtc.setTime(hours,minutes, seconds);

  // set the date ...

  // get compile date from the predefined macro - e.g. Oct 08 2022
  String d = __DATE__;

  String  s_month = d.substring(0,3);
  uint8_t day     = d.substring(4,6).toInt();
  uint8_t year    = d.substring(9,11).toInt();
```

```
  // parse the month to get as an integer (there are cleverer ways to
  // do this, but they are quite complicated!)
  uint8_t month = 0;
  if (s_month.equals("Jan"))
  {
    month = 1;
  }
  else if (s_month.equals("Feb"))
  {
    month = 2;
  }
  else if (s_month.equals("Mar"))
  {
    month = 3;
  }
  else if (s_month.equals("Apr"))
  {
    month = 4;
  }
  else if (s_month.equals("May"))
  {
    month = 5;
  }
  else if (s_month.equals("Jun"))
  {
    month = 6;
  }
  else if (s_month.equals("Jul"))
  {
    month = 7;
  }
  else if (s_month.equals("Aug"))
  {
    month = 8;
  }
  else if (s_month.equals("Sep"))
  {
    month = 9;
  }
  else if (s_month.equals("Oct"))
  {
    month = 10;
  }
  else if (s_month.equals("Nov"))
  {
    month = 11;
  }
  else if (s_month.equals("Dec"))
  {
    month = 12;
  }

  // set the date
  _rtc.setDate(day, month, year);
}
```

```
// get the date as a nicely formatted string
String MyRTC::get_rtc_date()
{
  // create a nicely formatted string
  char date_str[12];
  sprintf(date_str, "%02d/%02d/%02d",
          _rtc.getDay(), _rtc.getMonth(), _rtc.getYear());

  // return trhis character array wrapped in an arduino string
  return String(date_str);
}

// get the time as a nicely formatted string
String MyRTC::get_rtc_time()
{
  // create a nicely formatted string
  char time_str[12];
  sprintf(time_str, "%02d:%02d:%02d",
          _rtc.getHours(), _rtc.getMinutes(), _rtc.getSeconds());

  // return trhis character array wrapped in an arduino string
  return String(time_str);
}
```

We can then use this library in our code – see code listing 5.

Code listing 5: Code for the simple RTC example

```
/**
* 3_real_time_clock.ino
*
* this is a simple Arduino program to keep track of the time using
* a real-time clock (built into the MKR WiFi 1010) and display it
* on the serial monitor
*
* as the standard RTCZero library isn't very easy to initialise to
* the correct starting time / date I have written my own library to
* do this
*
* author: alex shenfield
* date:   24-10-2022
*/

// LIBRARY IMPORTS

// include my rtc library
#include "my_rtc.h"

// INITIAL VARIABLES

// create my rtc object
MyRTC rtc;

// CODE

// this method runs once (when the sketch starts)
void setup()
{
  // start serial comms and wait until its up and running
  Serial.begin(9600);
  while (!Serial)
  {
    delay(100);
  }
  Serial.println("RTC program running ...");

  // initialise the rtc with the compilation time and date
  rtc.set_rtc();
}

void loop()
{
  // print the time and date to the serial monitor
  Serial.print("current timestamp = ");
  Serial.print(rtc.get_rtc_time());
  Serial.print(" ");
  Serial.print(rtc.get_rtc_date());
  Serial.println();

  // delay for 10 seconds
  delay(10000);
}
```

Enter this program, upload it to the board, and make sure everything works.

Task 4

The second important part of data logging is storing the data! Whilst the Arduino chips have permanent EEPROM storage, this is only a couple hundred bytes – which is tiny compared to the storage on SD cards. Currently small SD cards are extremely cheap and easily available, so they are an obvious choice for long term storage of data.

The Arduino IDE provides a built-in SD library for reading data from and writing data to SD cards. It is built on the excellent sdfat library by Bill Greiman[5], and supports both FAT16 and FAT32 file systems. It is important to note that the SD library only support 8.3 file names (e.g. "***file.txt***") rather than the longer file names supported by modern operating systems.

In this task we are going to read data from a potentiometer and write it to the SD card in 1 second intervals. First build the system shown in Figure 13.

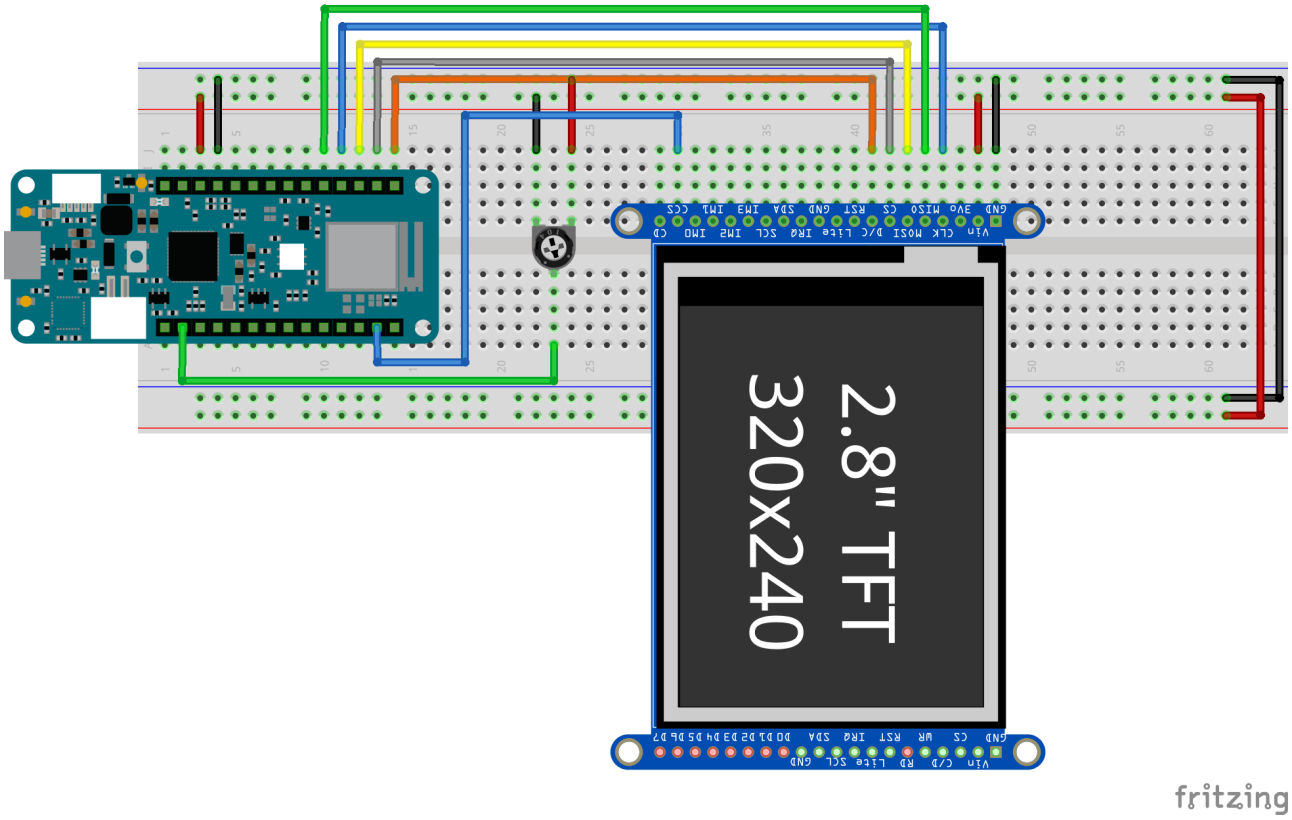

Figure 13 – Using the SD card slot on the breakout board for data logging

As with most of the other examples in this series of labs, our code will use the non-blocking delay mechanism introduced in the first lab to check whether a certain time period has elapsed and, if it has, open a file and print a formatted data string to it containing temperature and light readings. The code is shown in code listing 6, below.

5 https://github.com/greiman/SdFat

Code listing 6: Logging data to an SD card

```
/**
* 4_sd_card_data_logger.ino
*
* simple sketch to log the value from a potentiometer every
* 1000 ms
*
* author: alex shenfield
* date:   24-10-2022
*/

// LIBRARY IMPORTS

// include the spi and sd card libraries
#include <SPI.h>
#include <SD.h>

// INITIAL VARIABLES

// define the update interval
#define UPDATE_INTERVAL 1000

// PINS

// the potentiometer is connected to pin A0
const int pot_pin = A0;

// the sd card chip select (ccs) is connected to pin 4
const int chip_select = 4;

// TIMING

// timing variables ...
long previous_time = 0;

// CODE

// this method runs once (when the sketch starts)
void setup()
{
  // start serial comms and wait until its up and running
  Serial.begin(115200);
  while (!Serial)
  {
    delay(10);
  }
  Serial.println("SD card logger running ...");

  // see if the sd card is present and can be initialized
  if (!SD.begin(chip_select)) {
    return;
  }
  Serial.println("card initialised ...");

}
```

```
// this method loops continuously
void loop()
{
  // get the current time this time round the loop
  unsigned long current_time = millis();

  // if the set time interval has elapsed ...
  if (current_time - previous_time > UPDATE_INTERVAL)
  {
    // read the potentiometer
    int pot_val = analogRead(pot_pin);

     // print the value of the potentiometer to serial
    Serial.print("potentiometer value is: ");
    Serial.println(pot_val);

    // open the logfile
    File logfile = SD.open("log.txt", FILE_WRITE);

    // if the logfile is open the write the current time and
    // potentiometer value to it in csv format
    if (logfile)
    {
      logfile.print(current_time);
      logfile.print(",");
      logfile.println(pot_val);
      logfile.close();
    }
    else
    {
      Serial.println("error opening file!");
    }

    // set the previous time to the current time
    previous_time = current_time;
  }
}
```

Enter this code and make sure it works.

Sometimes the SD cards can be a bit fiddly so you may have to take it out and put it back in again if you get initialisation errors. You should see output on the Serial Monitor similar to that shown in Figure 14 (below).

Figure 14 – Data logger Serial Monitor window

If you remove the SD card from the breakout board and plug it into a computer, you should see a "LOG.TXT" file. If you open this in your favourite text editor you should see something like that shown in Figure 15.

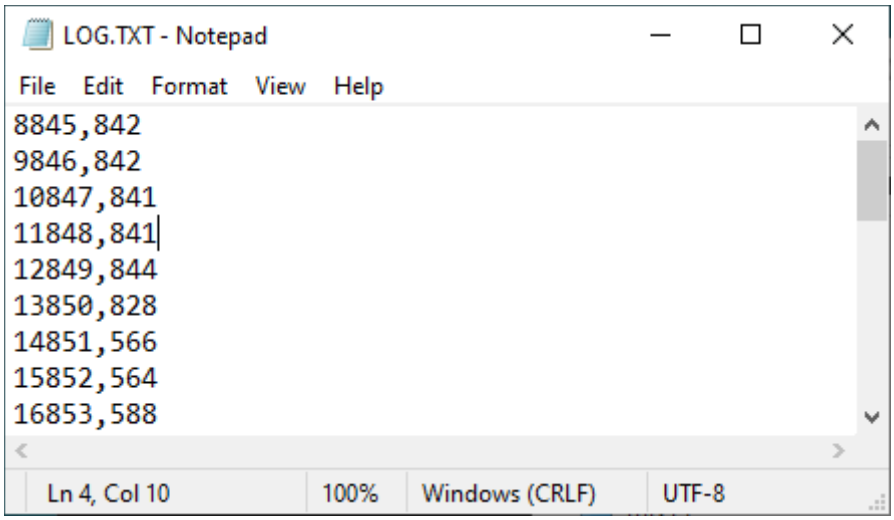

Figure 15 – The logged data from our potentiometer

Exercise

Now you are going to put together several of the things we have learnt in this and previous labs to build a sophisticated data logging system for environmental data. The system should have the following functionality:

1. Read temperature, humidity, and light data from appropriate sensors.
2. Display these values on the TFT display.
3. Log these values with RTC based timestamps to the SD card at an appropriate sampling interval.

You may also wish to display data on the Serial Monitor to help with debugging.

The system block diagram for this data logging system looks something like Figure 16 (below).

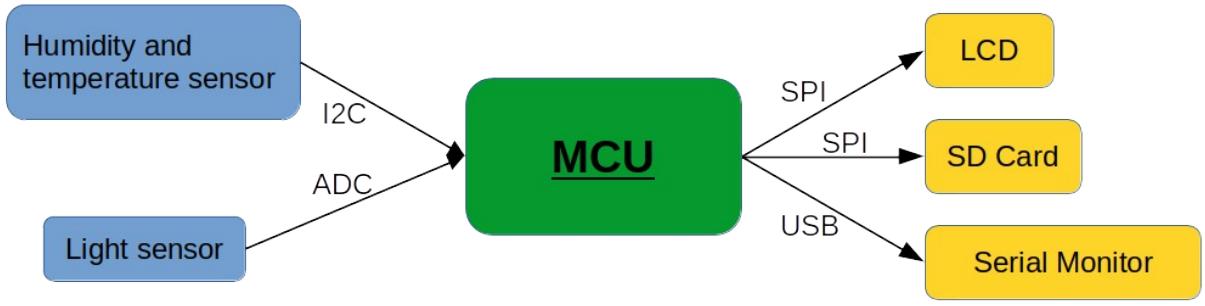

Figure 16 – Block diagram for the data logging system

My implementation of this system can be seen in Figure 17.

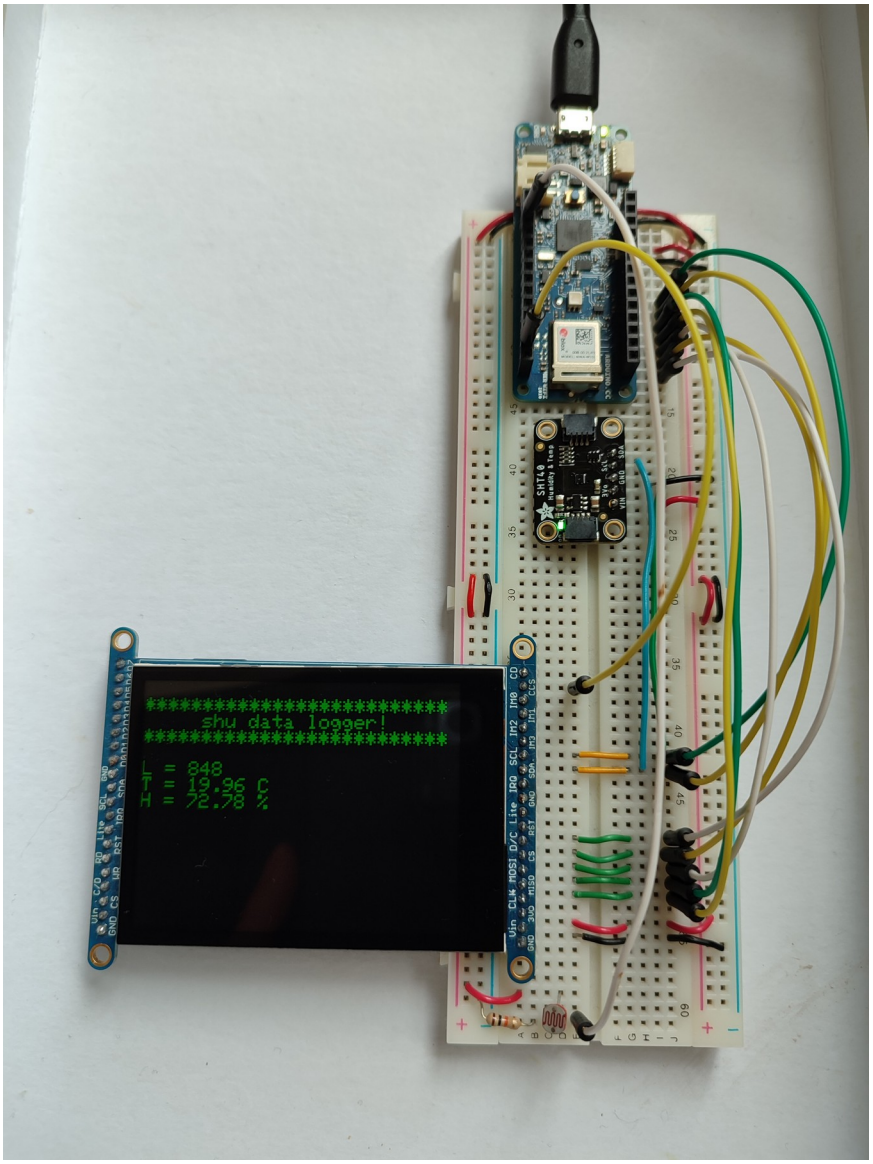

Figure 17 – My implementation of the environment data logger

Check list

Task	Completed
Task 1 – code	
Task 1 – exercise a	
Task 1 – exercise b	
Task 1 – exercise c	
Task 1 – exercise d	
Task 2 – code	
Task 2 – exercise a	
Task 2 – exercise b	
Task 2 – exercise c	
Task 3 – code	
Task 4 – code	
Task 4 – exercise	

Feedback