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

Activity ID		Activity Title			Laboratory Room No.	Level
Lab 103		Serial buses			4302	6
Semester	**Duration [hrs]**	**Group Size**	**Max Total Students**	**Date of approval**	**Lead Academic**	
2	4	1	25	09-24	Alex Shenfield	

Equipment (per student/group)

Number	Item
1	PC
1	Arduino kit

Learning Outcomes

	Learning Outcome
3	Select and apply appropriate embedded networking technologies for a given problem
4	Design, implement and test embedded networked devices

Risk Assessment

General risk assessment for Sheaf 4302 applies (see blackboard for more details).

Serial buses using 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 the basic 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 the use of serial bus protocols with the Arduino MKR WiFi 1010 board. Basic analog and digital inputs and outputs are fine for simple sensors (such as the potentiometers and LDRs we looked at in the last lab) but, as we've been discussing in the lectures, can be limited for more complex applications. In this lab we will look at using I2C and SPI to talk to sensors and displays so we can acquire and process more complex types of data.

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 SHT40 temperature and humidity sensor and 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, and a touch screen to allow us to interact with UI elements such as buttons. It also includes an SD card interface to allow us to log data points with time (but we will not be using that in this lab session).

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)!

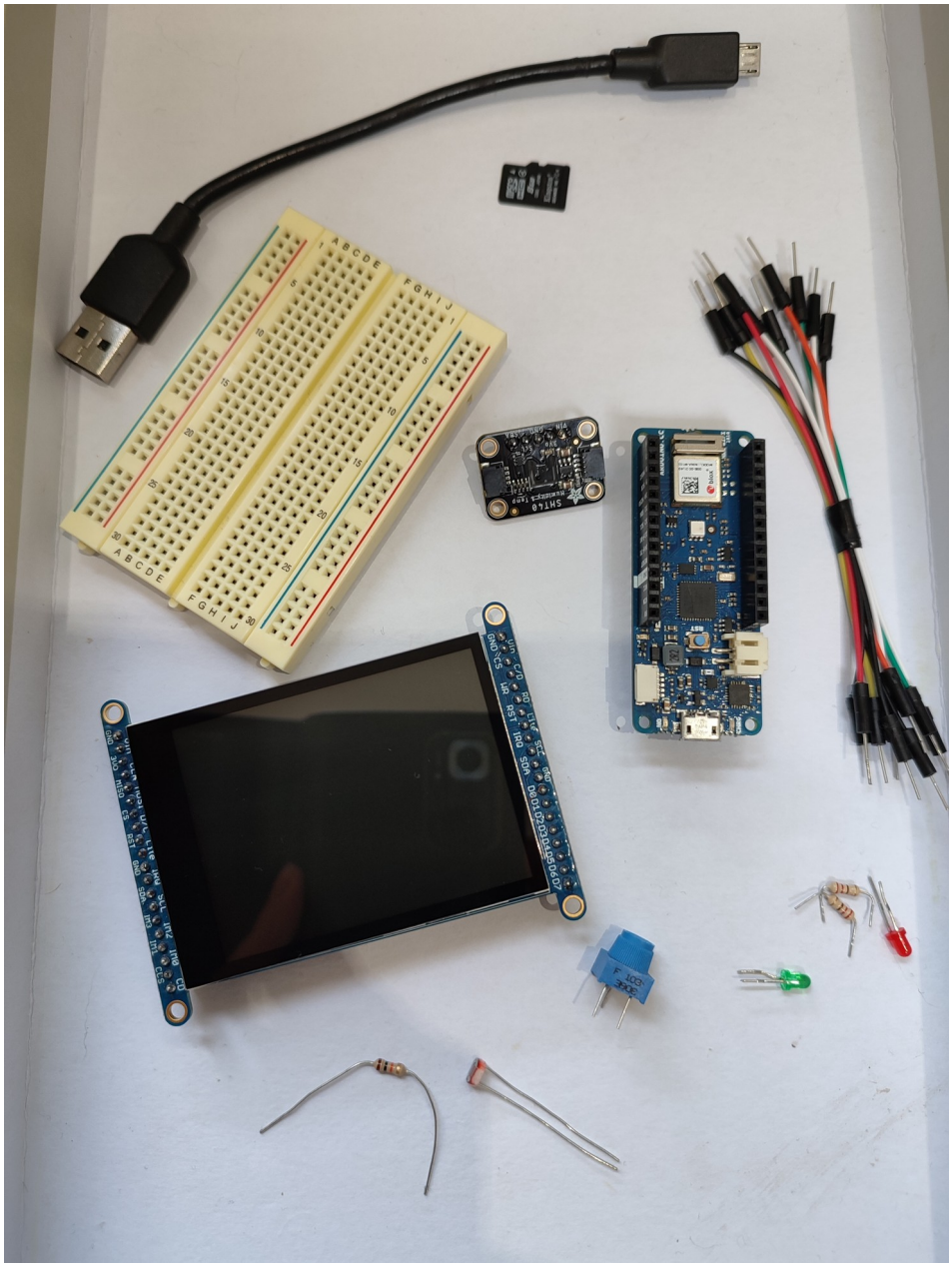


Figure 1 – The necessary equipment for this lab

Task 1

In this task we are going to read an I2C temperature and humidity sensor (the Adafruit Sensirion SHT40 Temperature & Humidity Sensor[2] - see Figure 2 and display the data in the Serial Monitor. The SHT40 has ±1.8% typical relative humidity accuracy from 25 to 75% and ±0.2 °C typical temperature accuracy from 0 to 75 °C. It is important to note that the I2C address for this sensor is 0x44 and cannot be changed[3].

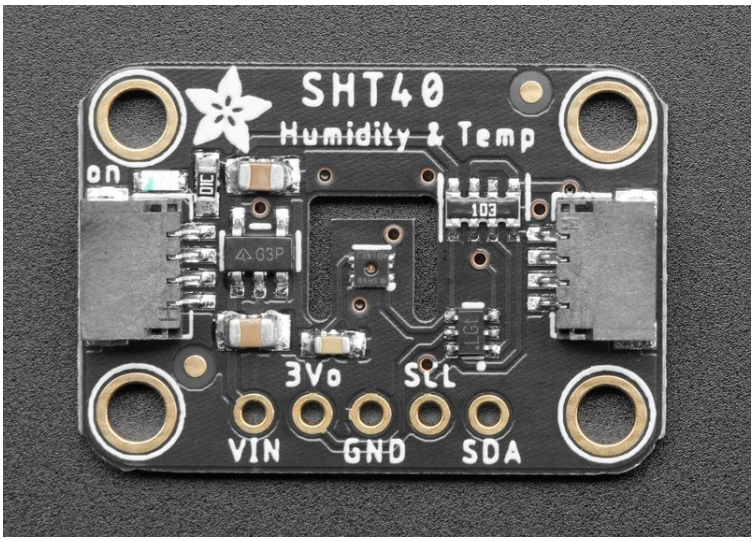


Figure 2 – The Adafruit Sensirion SHT40 Temperature & Humidity Sensor

The pin outs of this board are:

Pin	Description
VIN	This is the power pin – supply the same voltage as the logic level of your microcontrolle (i.e. 3.3V for the MKR WiFi 1010.
3Vo	This is the 3.3V output from the voltage regulator.
GND	Ground.
SCL	I2C clock line (note: there is an inbuilt 10K pull-up resistor on this line).
SDA	I2C data line (note: there is an inbuilt 10K pull-up resistor on this line too).

2 https://learn.adafruit.com/adafruit-sht40-temperature-humidity-sensor
3 So, as will be discussed the I2C lecture, if you have other I2C devices that use the same address you will probably run into issues!

Wire the sensor up to the Arduino MKR WiFi 1010 as shown in Figure 3. You can see on the side of the Arduino board there are dedicated pins for I2C communication (pins 11 and 12). Remember, I2C is a two wire protocol so all I2C devices are attached to common SCL and SDA connections.

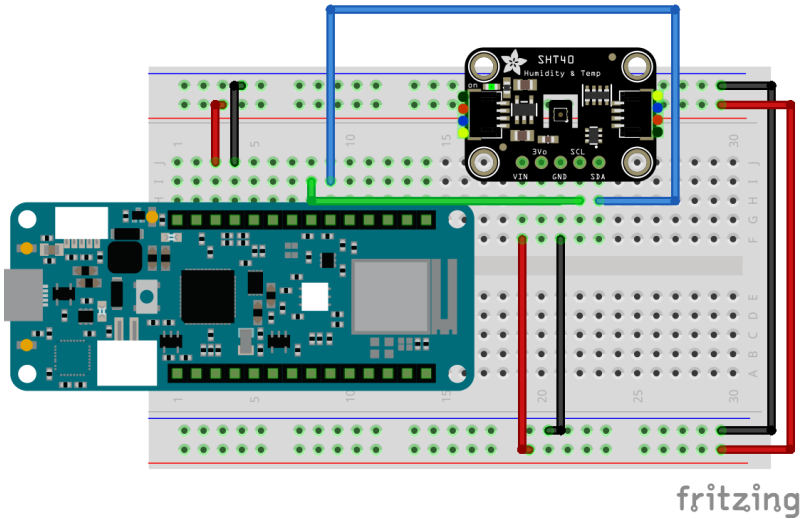
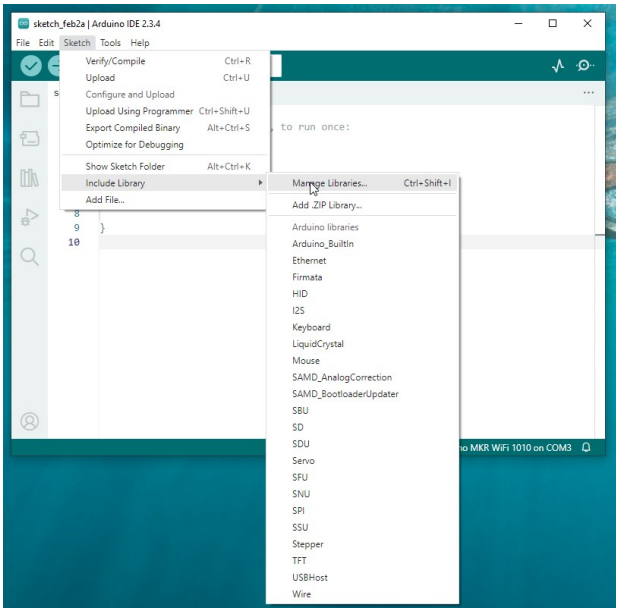
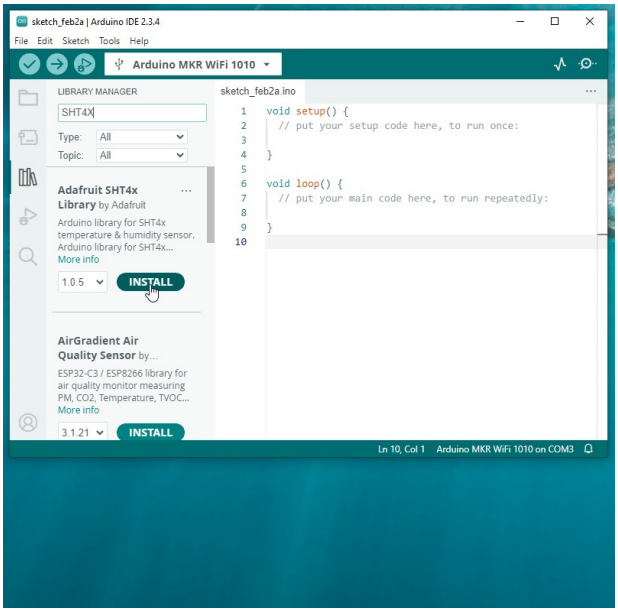


Figure 3 – SHT40 and Arduino MKR WiFi 1010 circuit

Before programming the board, we will need to install the Adafruit SHT4X Library using the Arduino IDE Library Manager (see Figure 4). We do this is the same way as adding the EasyButton library in the previous lab.



a) Selecting “Manage Libraries ...”



b) Searching for the “SHT4X” library and installing it

Figure 4 – Installing the Adafruit SHT4X Library

You will probably then see a message about dependencies for the Adafruit SHT4X Library (see Figure 5). Just click “Install all” to install these alongside the Adafruit SHT4X Library.

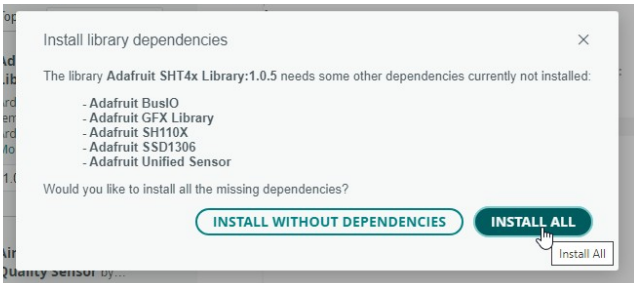


Figure 5 – Installing library dependencies

Now we need to write some code to configure and read the sensor. The SHT40 has several levels of precision / repeatability we can configure when reading the sensor, and a variable power heating element (useful for removing condensed water from the surface of the sensor to ensure more accurate readings in high humidity environments). See code listing 1 for a simple program to configure and read the sensor for normal ambient conditions.

Code listing 1: Code to configure and read the SHT40 sensor and display the values on the Serial Monitor

```
/*
 * 1_interfacing_with_sensors.ino
 *
 * this is a simple Arduino program to read from an i2c sensor (in this
 * case an SHT40 temperature and humidity sensor) and write the sensor
 * reading to the serial monitor
 *
 * this program uses pins 11 and 12 for SDA and SCL respectively (these
 * are the default i2c pins on the Arduino MKR WiFi 1010)
 *
 * author: alex shenfield
 * date:   2024-11-12
 */

// LIBRARY IMPORTS

// we will use the adafruit SHT4X library to simplify use of this
// sensor
#include "Adafruit_SHT4x.h"

// INITIAL VARIABLES

// create the sensor object
Adafruit_SHT4x sht40 = Adafruit_SHT4x();

// CODE

// this method runs once (when the sketch starts)
void setup()
{
  // initialise serial communication at 115200 bits per second, wait for
  // the serial port to become available, and print a start up message
  Serial.begin(115200);
  while (!Serial)
  {
    delay(1);
  }
  Serial.println("i2c sensor example");

  // start the SHT40 sensor
  if (!sht40.begin())
  {
    // if it doesn't connect, print an error message and wait forever
    Serial.println("couldn't find SHT40 sensor - check connections!");
    while (true)
    {
      delay(1);
    }
  }

  // print the serial number of the sensor (for information)
  Serial.println("found SHT40 sensor");
  Serial.print("serial number 0x");
  Serial.println(sht40.readSerial(), HEX);

  // configure the sensor to have high precision (i.e. high repeatability)
  // and heater off - see lab sheet for description of when using the
  // heater is useful
  sht40.setPrecision(SHT4X_HIGH_PRECISION);
  sht40.setHeater(SHT4X_NO_HEATER);

}
```

```
// this methods loops continuously
void loop()
{
  // initialise structures to contain the sensor readings
  sensors_event_t humidity;
  sensors_event_t temp;

  // read the sensor (and keep track of how long it takes)
  long timestamp = millis();
  sht40.getEvent(&humidity, &temp);
  timestamp = millis() - timestamp;

  // print temperature
  Serial.print("temperature: ");
  Serial.print(temp.temperature);
  Serial.println(" degrees C");

  // print humidity
  Serial.print("humidity: ");
  Serial.print(humidity.relative_humidity);
  Serial.println("%");

  // print the sensor read time
  Serial.print("reading took ");
  Serial.print(timestamp);
  Serial.println(" ms");

  // we'll use a simple 1s delay here because our loop is simple - if
  // we wanted to do anything more complicated in our loop, we'd be
  // better off not using delays (because they mean our program can't
  // do anything else)
  delay(1000);
}
```

Enter this program, upload it to the board, and make sure everything works. When you access the Serial Monitor, you should see data displayed similar to that shown in Figure 6.

Figure 6 – Display of temperature and humidity data

Exercises

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

a) Add some indicator LEDs to light up when the temperature and humidity values go above some preset threshold values.

b) Add two potentiometers to set these threshold values. The Arduino MKR WiFi 1010 analog to digital convertors read from 0 – 1023, so you will have to remap these values to a sensible range (e.g. 5 – 30 for temperature and 0 – 100 for relative humidity). You can do this by using the "map" command in the Arduino language:

https://www.arduino.cc/reference/en/language/functions/math/map/

c) Now add the "smart" light sensor functionality you developed for task 4a) in lab 102 to your system. You now have the basis of a building environment monitoring system!

Task 2

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[4] – 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 7.

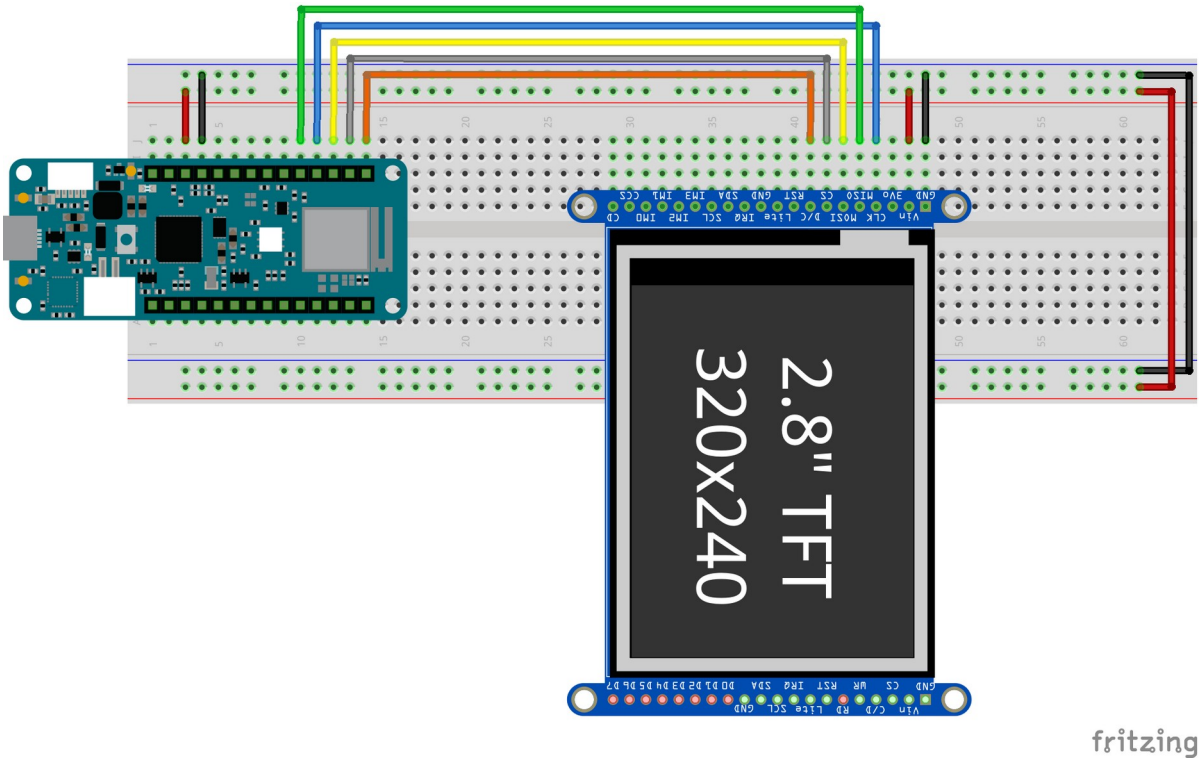


Figure 7 – Arduino MKR WiFi 1010 connected to the Adafruit 2.8" TFT LCD with capacitive touch screen

4 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 8 shows the LCD display showing a simple “hello world” message and uptime counter (code for this application is provided in code listing 2). This uptime counter is updated every second.

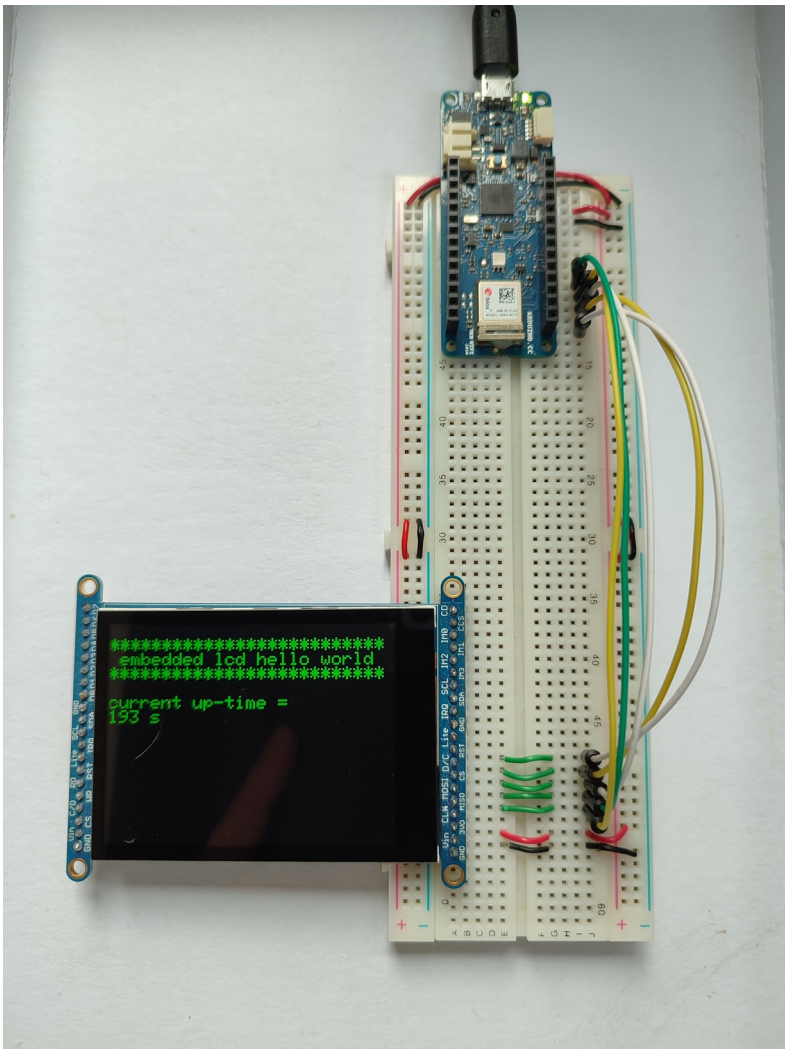


Figure 8 – The LCD “hello world” program in action

Code listing 2: Code for the simple lcd hello world program

```
/*
 * 2_using_a_display.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:   2024-11-12
 */

// 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
#define TFT_RS 5

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

// 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 9 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 9 – 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[5]. I would **highly** recommend using pseudocode or other algorithm design methodologies before actually writing the code! Figure 10 shows an example of what this might look like.

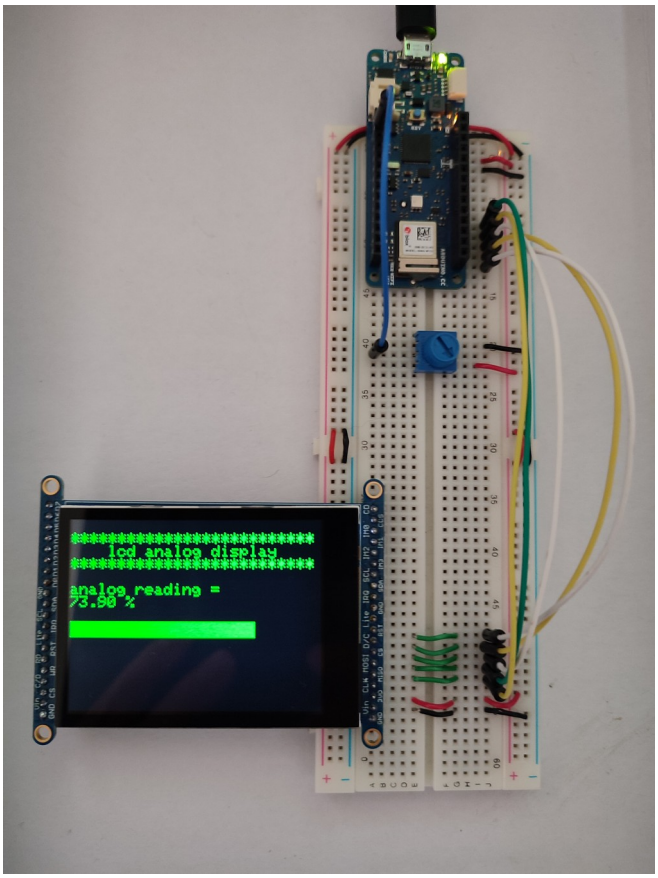


Figure 10 – An example bar graph display on the Arduino MKR WiFi 1010 and Adafruit 2.8" TFT display

5 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 3

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 11 (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.

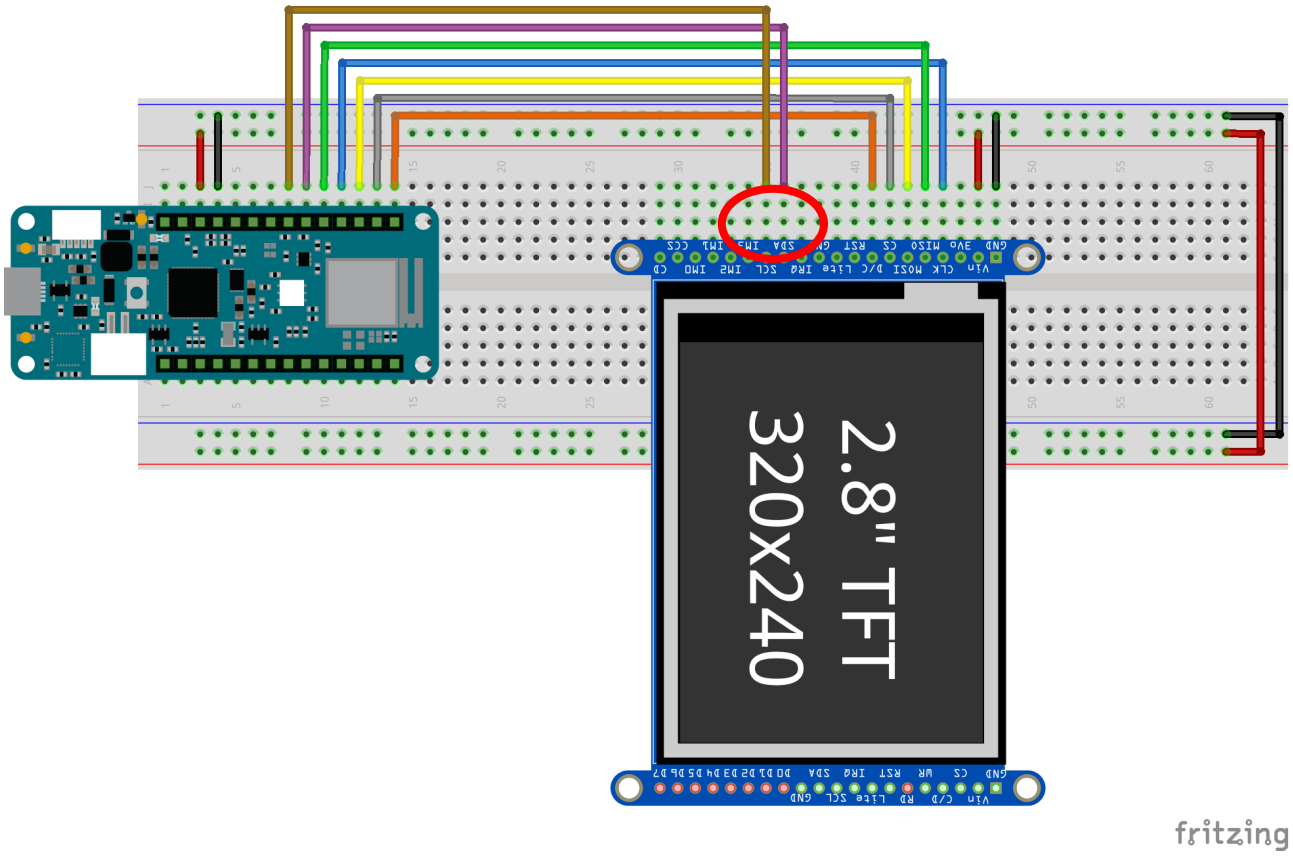


Figure 11 – Fritzing breadboard schematic of the TFT LCD breakout board with the touch screen enabled

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 12).

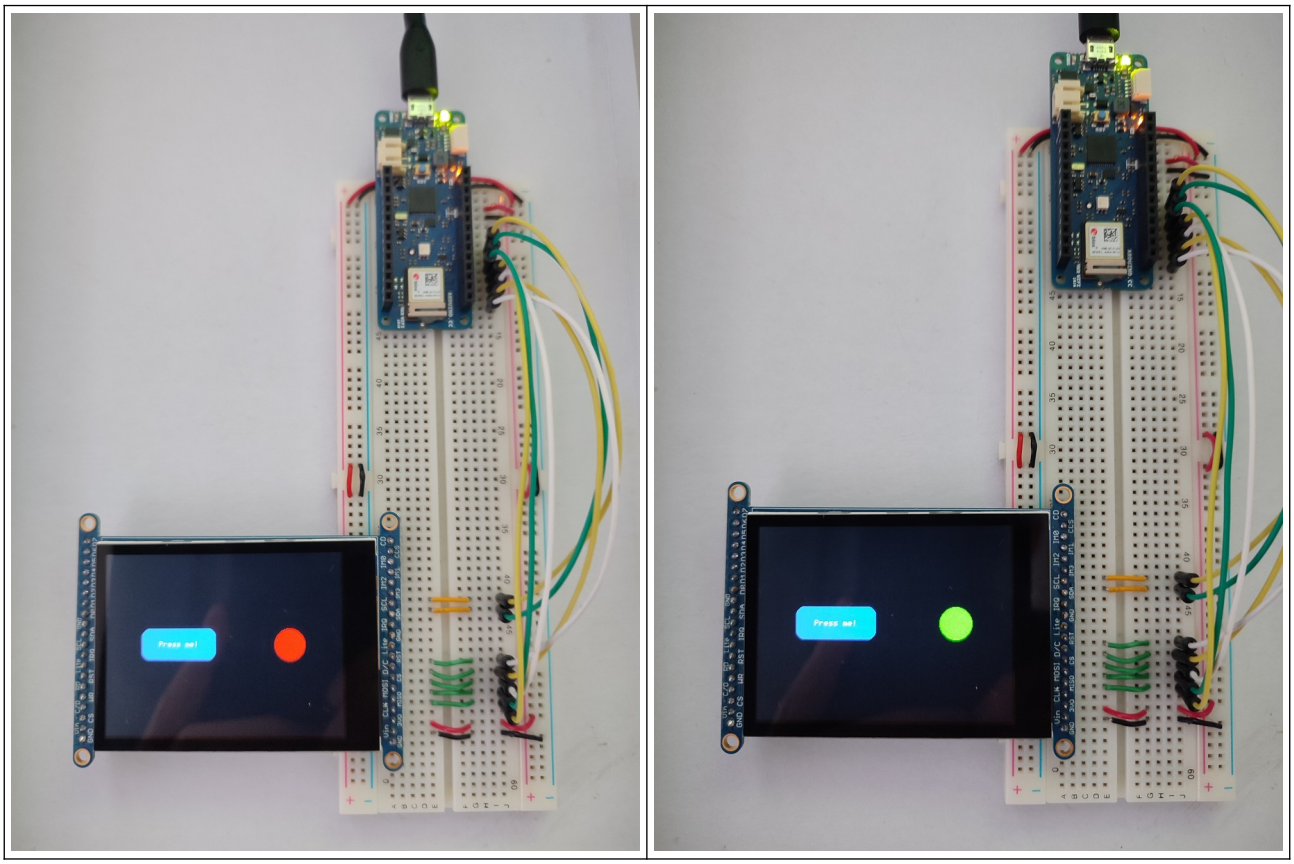


Figure 12 – The touchscreen example

Code listing 3 provides the code for this “virtual led” example.

Code listing 3: Code for the touch screen virtual led program

```
/*
 * 3_using_the_touchscreen.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:   2024-11-12
 */

// 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;
    }
  }
}
```

```
// UTILITY METHODS

// print some simple diagnostic information about the TFT LCD (this is optional
// but can help debug problems – e.g. the lcd not being 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.

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

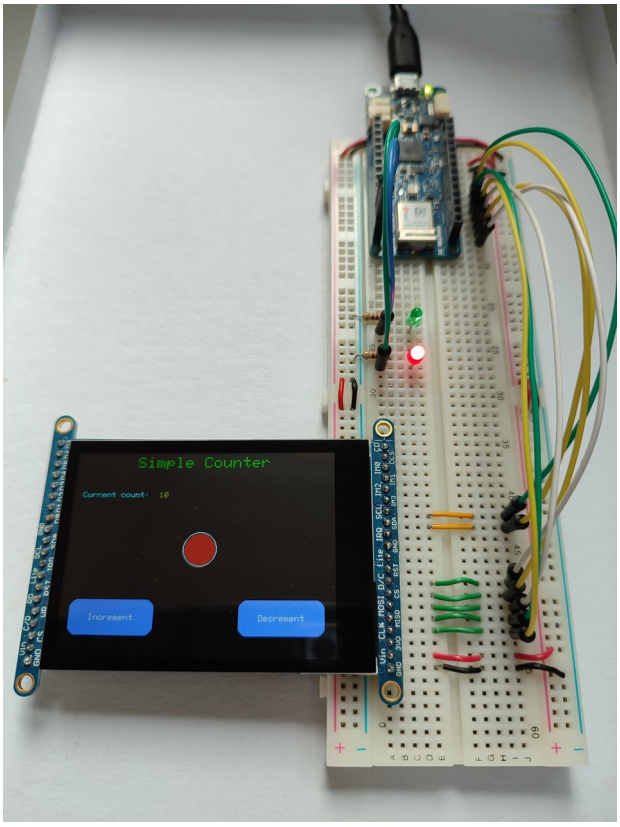


Figure 13 – Touchscreen counter example

Check list

Task	Completed
Task 1 – Code	
Task 1 – Exercise a	
Task 1 – Exercise b	
Task 1 – Exercise c	
Task 2 – Code	
Task 2 – Exercise a	
Task 2 – Exercise b	
Task 2 – Exercise c	
Task 2 – Exercise d	
Task 3 – Code	
Task 3 – Exercise a	
Task 3 – Exercise b	
Task 3 – Exercise c	

Feedback