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

Activity ID		Activity Title			Laboratory Room No.	Level
Lab 102		Advanced I/O			4302	6
Term	**Duration [hrs]**	**Group Size**	**Max Total Students**	**Date of approval/review**	**Lead Academic**	
1	4	2	20	09-22	Alex Shenfield	

Equipment (per student/group)

Number	Item
1	STM32F7 discovery board lab 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

Seeing the world, acquiring data, and reacting to external conditions

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 the laboratory sessions for this module you are going to be introduced to the STM32F7 discovery board – a powerful ARM Cortex M7 based microcontroller platform capable of prototyping advanced embedded systems designs. The STM32F7 discovery board includes advanced functionality such as Ethernet connectivity, UART over the USB connection, an LCD screen, and a micro-SD slot. Appendix B shows the various pins that are broken out from the STM32F7 discovery board (onto the Arduino form factor header).

This laboratory session will focus on advanced input and output features with the STM32F7 discovery board. As mentioned in the last lab sheet, any kind of embedded microcontroller is only really useful if it has some ability to see the world – whether as simple as taking input from switches and potentiometers or as complicated as reading and reacting to accelerometer data. To do this we will need to integrate sensors into our system.

As well as taking input from real world sensors, we will look in this lab sheet at the display of data via the LCD screen that forms part of our STM32F7-discovery evaluation kit. The display of data is an important feature in any embedded system.

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.cs.indiana.edu/~geobrown/book.pdf[1]
- https://visualgdb.com/tutorials/arm/stm32/
- http://www.keil.com/appnotes/files/apnt_280.pdf
- https://developer.mbed.org/platforms/ST-Discovery-F746NG/

1 Note: this book is for a slightly different board – however, much of the material is relevant to the STM32F7 discovery

Methodology

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

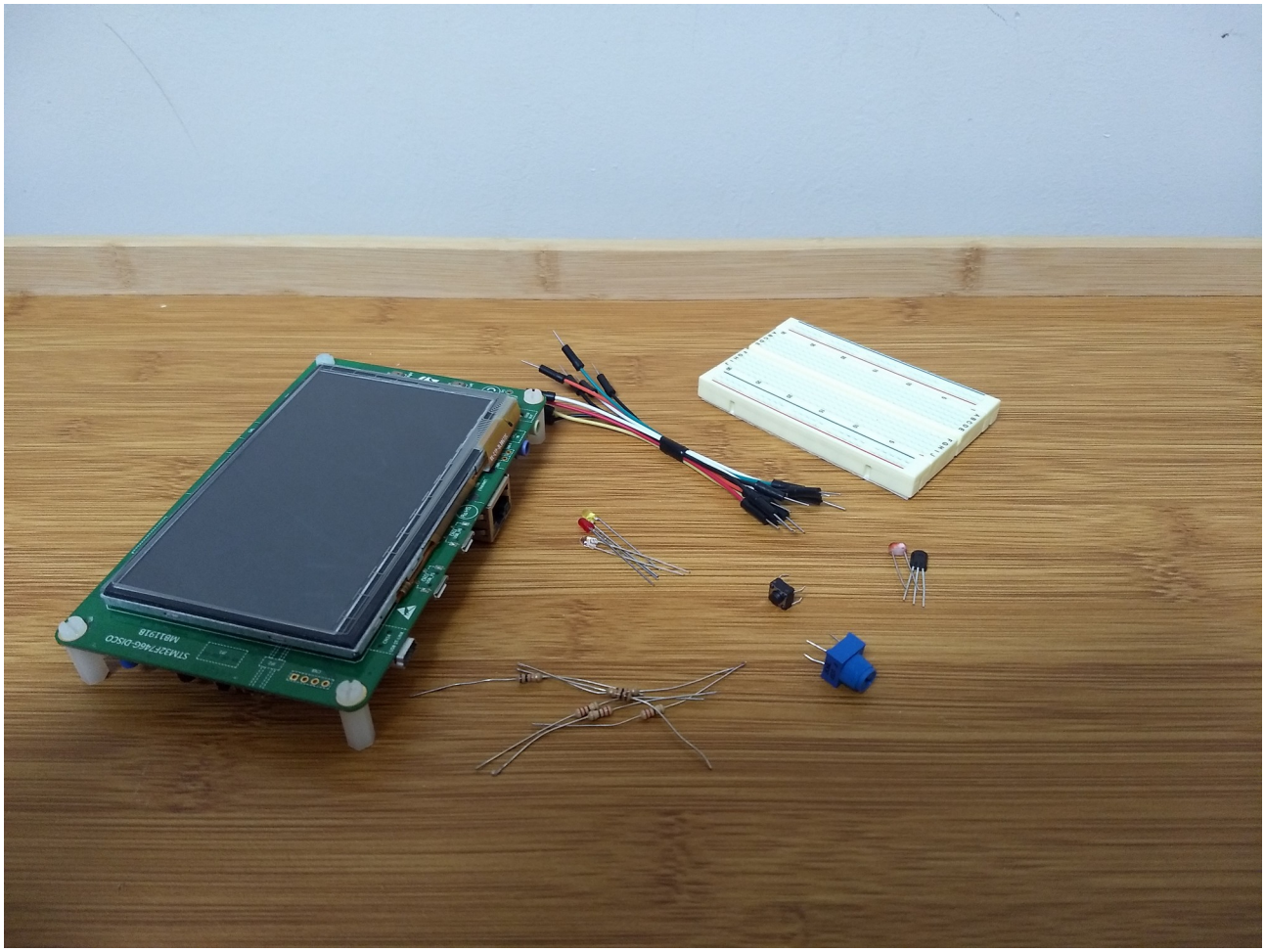


Figure 1 – The necessary equipment for this lab (don't worry if you have LEDs of a different colour or different potentiometers)

Task 1

The STM32F7 discovery board lab kits on the benches have a 4.5” touch screen LCD built in to the evaluation board. This touch screen LCD allows us to draw shapes, display data as text, and even to render bitmaps!

Figure 2 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.

Run the code provided (the project can be downloaded from GitHub) and verify it works. Make sure you go through the code in detail to understand what it is doing.

Figure 2 – The LCD “hello world” program in action

Code listing 1:

```
/*
 * main.c
 *
 * this is the main lcd application
 *
 * author:  Dr. Alex Shenfield
 * date:    05/10/2021
 * purpose: 55-604481 embedded computer networks : lab 102
 */

// include the basic headers and hal drivers
#include "stm32f7xx_hal.h"

// include the shu bsp libraries for the stm32f7 discovery board
#include "pinmappings.h"
#include "clock.h"
#include "stm32746g_discovery_lcd.h"

// LCD DEFINES

// define a message boarder (note the lcd is 28 characters wide using Font24)
#define BOARDER   "****************************"

// specify a welcome message
const char * welcome_message[2] =
{
  "*   Hello LCD World!   *",
  "*    Welcome to SHU    *"
};
```

```
// CODE

// this is the main method
int main()
{
  // we need to initialise the hal library and set up the SystemCoreClock
  // properly
  HAL_Init();
  init_sysclk_216MHz();

  // initialise the lcd
  BSP_LCD_Init();
  BSP_LCD_LayerDefaultInit(LTDC_ACTIVE_LAYER, SDRAM_DEVICE_ADDR);
  BSP_LCD_SelectLayer(LTDC_ACTIVE_LAYER);

  // set the background colour to blue and clear the lcd
  BSP_LCD_SetBackColor(LCD_COLOR_BLUE);
  BSP_LCD_Clear(LCD_COLOR_BLUE);

  // set the font to use
  BSP_LCD_SetFont(&Font24);

  // print the welcome message ...
  BSP_LCD_SetTextColor(LCD_COLOR_WHITE);
  BSP_LCD_DisplayStringAtLine(0, (uint8_t *)BOARDER);
  BSP_LCD_DisplayStringAtLine(1, (uint8_t *)welcome_message[0]);
  BSP_LCD_DisplayStringAtLine(2, (uint8_t *)welcome_message[1]);
  BSP_LCD_DisplayStringAtLine(3, (uint8_t *)BOARDER);

  // delay a little ...
  HAL_Delay(5000);

  // display an "uptime" counter
  BSP_LCD_DisplayStringAtLine(5, (uint8_t *)"Current uptime =");
  int counter = 0;
  while(1)
  {
    // format a string based around the uptime counter
    char str[20];
    sprintf(str, "%d s", counter++);

    // print the message to the lcd
    BSP_LCD_ClearStringLine(6);
    BSP_LCD_DisplayStringAtLine(6, (uint8_t *)str);

    HAL_Delay(1000);
  }
}
```

Task 2

Previously we have read data from a potentiometer and used it to control the rate at which an LED flashed. Now you should add a potentiometer to the system and output the potentiometer value on the LCD display. You will have to use the sprintf()[2] function from the c standard io library to create the formatted string for writing to the LCD. Code listing 2 (below) provides an example of how to use this function (the eagle eyed amongst you will notice that this is pretty much the same as the code for displaying the uptime counter in the last task).

Code listing 2:

```
// include the stdio library
#include <stdio.h>

// <snip ...>

// main loop goes here ...

  // format a string based around the adc value and print to lcd
  char str[12];
  sprintf(str, “ADC = %4d”, adc_val);
  BSP_LCD_DisplayStringAtLine(6, (uint8_t *)str);

// main loop ends here ...
```

Write your program!

2 https://www.tutorialspoint.com/c_standard_library/c_function_sprintf.htm

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

1. 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 a percentage for the potentiometer value.

2. 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.

3. 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 3 shows an example of what this might look like.

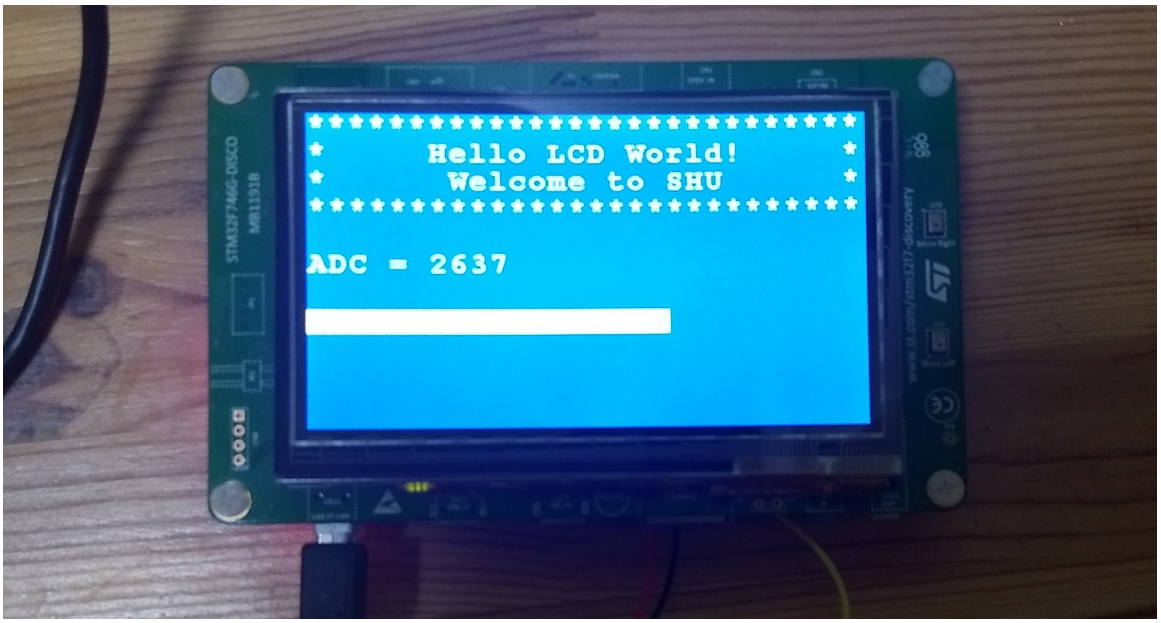


Figure 3 – An example bar graph display on the stm32f7 discovery board LCD

3 The *stm32746g_discovery_lcd.c* library provides a:

```
BSP_LCD_FillRect(uint16_t xpos, uint16_t ypos, uint16_t Width, uint16_t Height);
```

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

Task 3

You are now required to use the integrated circuit temperature sensors and light dependents resistors supplied to measure environmental data.

The temperature sensors we will be using are Analog Device TMP36s (see Figure 4) and extracts from the data sheet are provided in Appendix A (the full datasheet is provided on blackboard). Ensure that you insert these temperature sensors the correct way round.

They get very hot otherwise!!

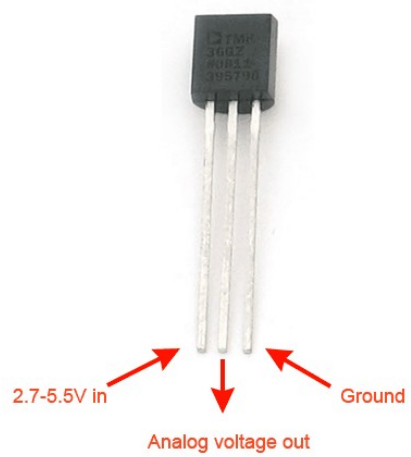


Figure 4 – A TMP36 temperature sensor

These sensors use a solid-state technique to determine the temperature. That is to say, they don't use mercury (like old thermometers), bimetalic strips (like in some home thermometers or stoves), nor do they use thermistors (temperature sensitive resistors). Instead, they use the fact as temperature increases, the voltage across a diode increases at a known rate. They amplify this voltage change by a precise amount to generate an analog signal that is directly proportional to the temperature.

Because these sensors have no moving parts, they are precise, never wear out, don't need calibration, work under many environmental conditions, and are consistent between sensors and readings. Moreover they are very inexpensive and quite easy to use.

As you can see from Figure 4 we get an analog voltage out (much like using a potentiometer). We will then have to convert that into a temperature using information from the data sheet (see Appendix A).

If you are unsure how to read the data sheet – ask on the blackboard discussion boards!

As well as the temperature sensor (above), we are also going to use light dependent resistors to obtain important information about the state of our environment (i.e. how dark it is!). Figure 5 shows one of the variants of light dependent resistor stocked by the university.

Figure 5 – An LDR

You might notice that the LDR only has two legs – to get an output voltage corresponding to the light level you need to set it up as a voltage divider (see Figure 6).

Figure 6 shows a breadboard schematic for the circuit. Note, you can choose input pins other than PA0 or PF10 – but you should make sure that it works with the ADC[4]. The easiest way to do this is to stick to the pins that are broken out as A0-A5 on the Arduino form factor header (shown in Appendix B).

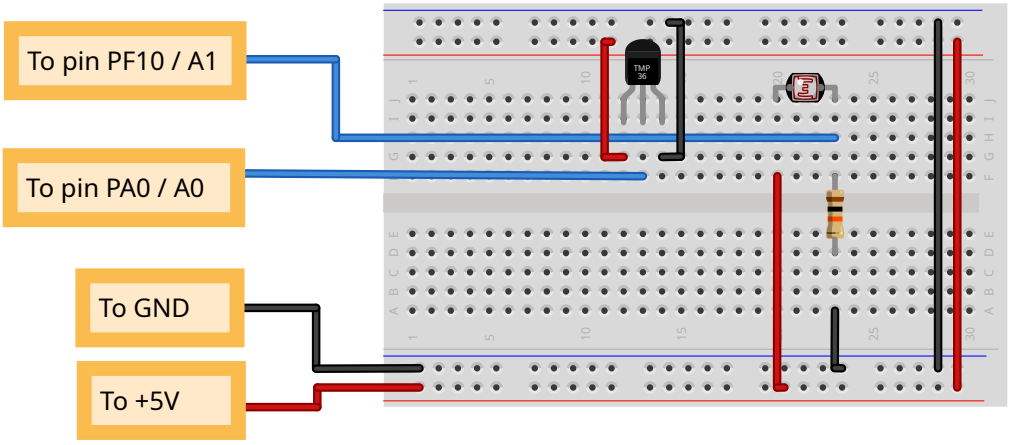


Figure 6 – The environment sensing circuit

Now write a program to read the temperature sensor and light dependent resistor and display this information on the LCD screen. Your program should display temperature in degrees Celsius (i.e. after conversion using the formula on the datasheet).

4 To find out which pins work with the ADC we have to look at the stm32f7 data sheet - "DM00166116.pdf" in table 10 from pg 53 onwards - available on Blackboard

An aside: circuit debugging ...

You may have noticed during Task 3 that there are a few issues with calculating the temperature values in your program:

1. the temperature values produced by your code seem to bear little relation to the true temperature values (e.g. sometimes we see negative or very low temperature values)
2. when the light readings change, the temperature readings also seem to change (sometimes only slightly, but sometimes by a lot)

These issues occur due to the input impedance of the ADC on the STM32F7 chip – to get accurate readings, the impedance of the source must be much much lower than the impedance of the ADC, otherwise the impedance of the ADC significantly effects the voltage measured[5].

We can confirm this by using a multimeter to measure the actual voltage of the temperature sensor, and compare it to the reading from the ADC (see Figures 7 and 8).

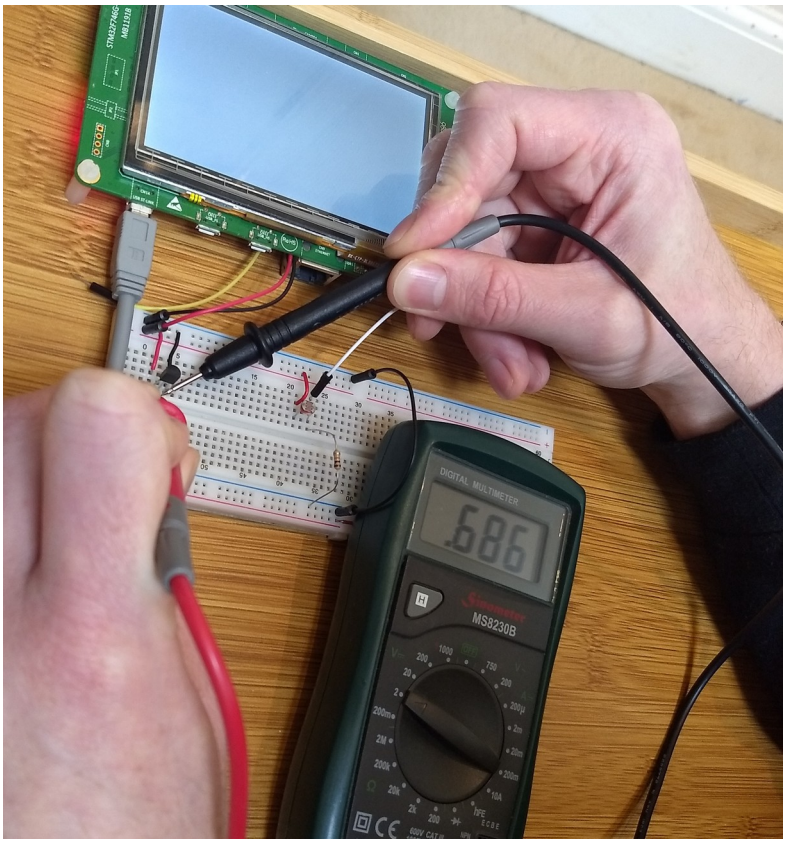


Figure 7 – Reading the temperature sensor voltage with a multimeter

5 This is why the impedance of multimeters and oscilloscopes is generally very high – because otherwise the act of measuring the voltage source will effect the measurement itself.

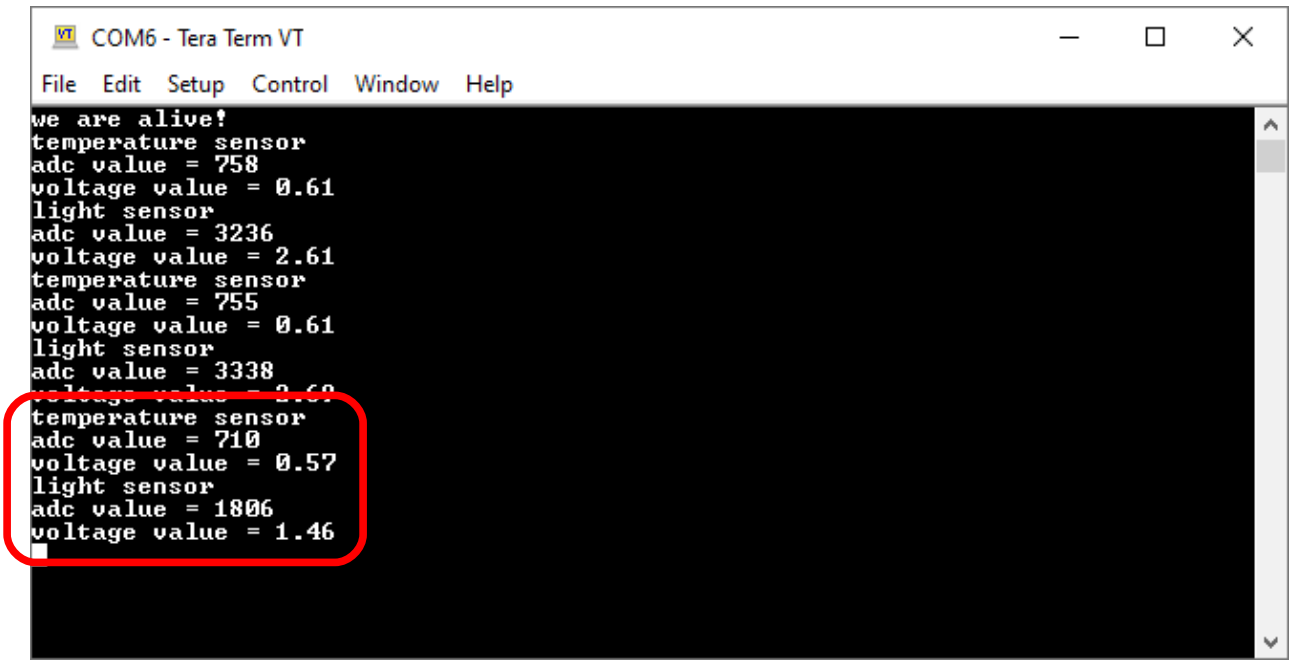


Figure 8 – ADC readings for the temperature sensor (the highlighted area is where the light sensor reading changed)

You can see that the readings from the ADC in Figure 8 (even when the light sensor reading isn't changed) are approximately 75mV different from that measured by the multimeter (in Figure 1)! This equates to 7.5°C in the temperature conversion!

The simplest fix to this issue is to use a op amp buffer on the output signal of the sensor to ensure that the source impedance (i.e. the sensor signal to the ADC) is as low as possible. I use LM358 dual op amp ICs (which can be bought off eBay for less than £2 for 5 – or ask in the technicians office if they have any kicking around). Figure 9 shows the breadboard schematic diagram, and Figure 10 shows the circuit schematic. Figure 11 gives the pin out for the LM358 chip that I am using here (taken from the datasheet).

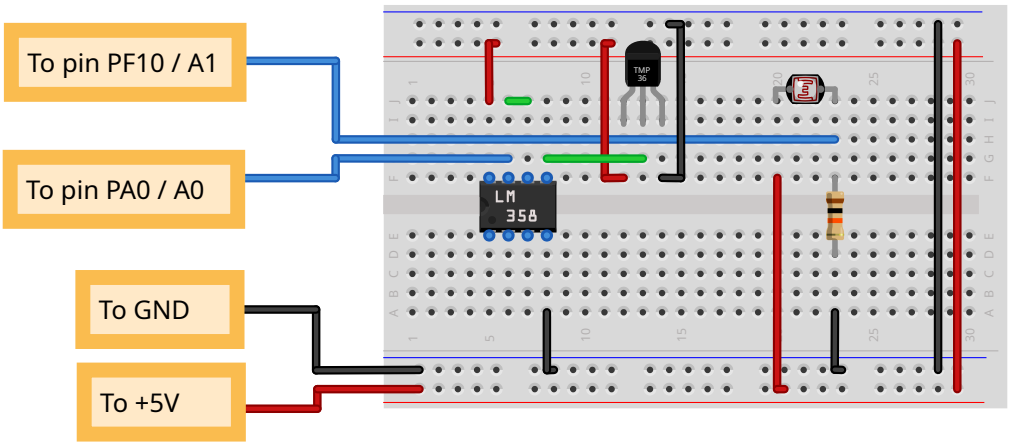


Figure 9 – Improving analog sensor reading using an op amp buffer

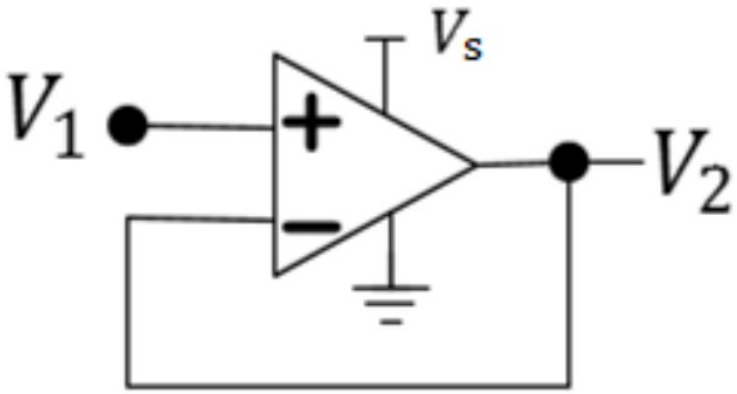


Figure 10 – Op amp buffer / voltage follower circuit

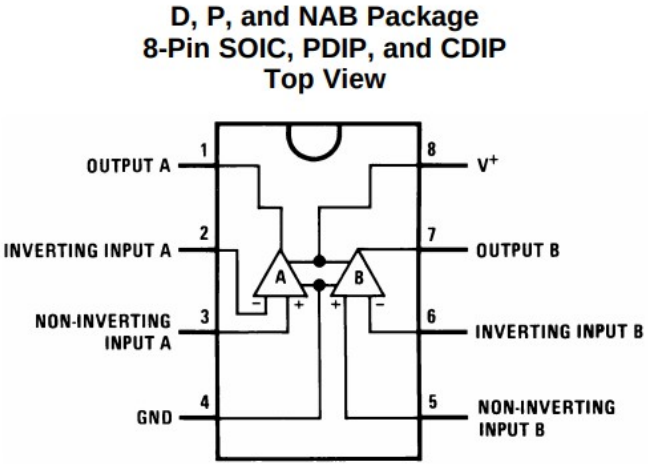


Figure 11 – The LM358 pin out

Figure 12 shows the full layout of my working system.

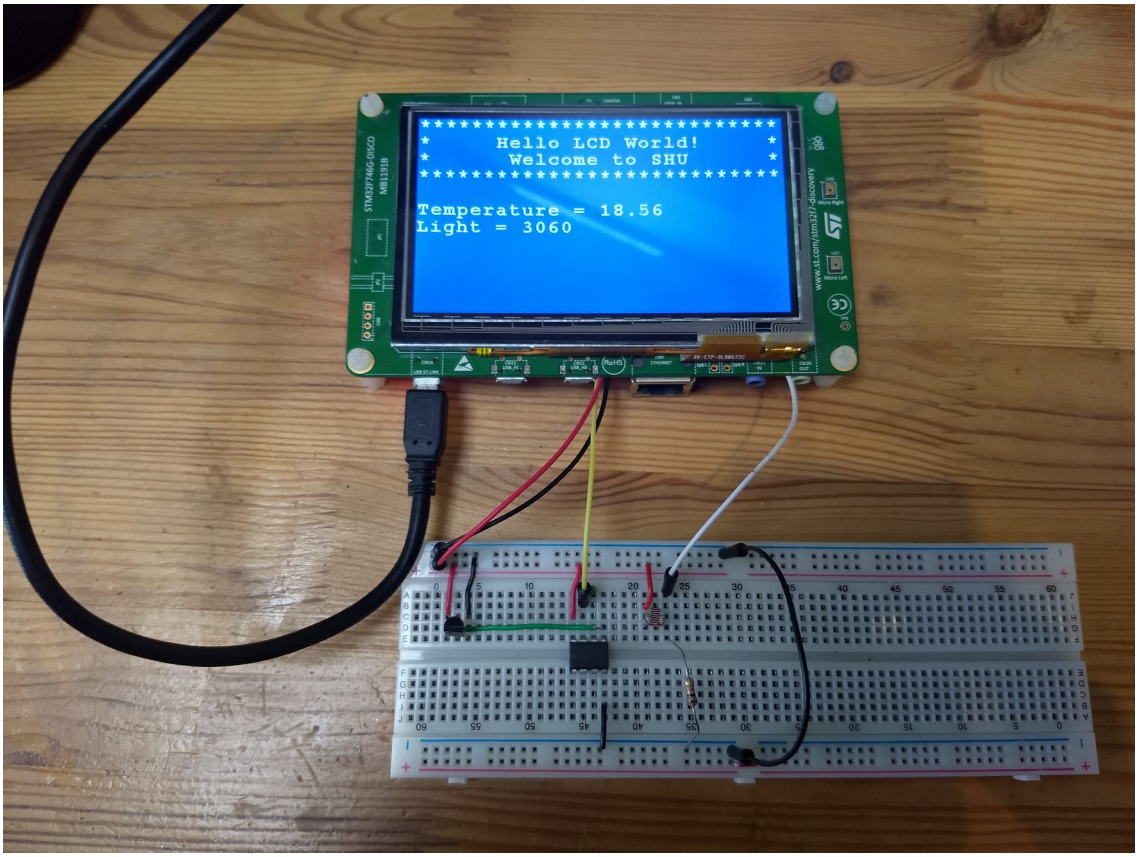


Figure 12 – Environment sensor circuit with an op amp buffer

As you can see the accuracy of the temperature reading is much improved! And as an added bonus, it is much less sensitive to changes in the other analog inputs.

Task 4

In any kind of data logging system it is often very useful to include timestamps to indicate when data was acquired. Whilst we can use elapsed time (as in Task 1 on this lab sheet), it is better if we can use a real time value. Fortunately, the STM32F7 chips include built in RTC (real-time clock) functionality! I have added a simple wrapper around this RTC functionality to the SHU board support package to enable:

1) RTC initialisation
2) RTC date and time configuration
3) Reading the date and time from the RTC

Code listing 3 shows the main program for using the RTC. A full project is available as part of lab 102 on GitHub.

Code listing 3:

```
/*
* main.c
*
* this is the main rtc calendar application (this code is dependent on the
* extra shu libraries such as the pinmappings list, the clock configuration
* library, the rtc library, and the gpio library)
*
* author:    Dr. Alex Shenfield
* date:      05/10/2021
* purpose:   55-604481 embedded computer networks : lab 102
*/

// include the c standard io functions
#include <stdio.h>

// include the hal drivers
#include "stm32f7xx_hal.h"

// include the shu bsp libraries for the stm32f7 discovery board
#include "pinmappings.h"
#include "clock.h"
#include "adc.h"
#include "gpio.h"
#include "real_time_clock.h"

// DECLARATIONS

// stdout_init is defined in our configuration file (as part of the ARM
// Compiler user code template
extern int stdout_init(void);
```

```
// CODE

// this is the main method
int main()
{
  // we need to initialise the hal library and set up the SystemCoreClock
  // properly
  HAL_Init();
  init_sysclk_216MHz();

  // set our current date ...
  rtc_date_t date;
  date.day = RTC_WEEKDAY_TUESDAY;
  date.day_date = 0x05;
  date.month = RTC_MONTH_OCTOBER;
  date.year = 0x21;

  // ... and time
  rtc_time_t time;
  time.hour = 0x10;
  time.minute = 0x52;
  time.second = 0x00;

  // initialise the uart and rtc (and force configuration of date and time)
  stdout_init();
  init_rtc(date, time);
  config_datetime(date, time);

  // print an initial status message
  printf("we are alive!\r\n");

  // loop forever ...
  while(1)
  {
    // get the date and time and print to the terminal
    uint8_t time[50] = {0};
    uint8_t date[50] = {0};
    get_datetime(time, date);
    printf("%s %s\r\n", date, time);

    // pause for ~1 sec
    HAL_Delay(1000);
  }
}
```

Build and run the code provided (the project can be downloaded from GitHub) and verify it works.

Now adapt your program from Task 3 to add RTC timestamps and serial port logging of the environment data.

Task 5

As well as a 480 x 271 pixel graphical LCD, the display on the STM32F7 discovery board is also multi-touch sensitive (using a capacitive touch screen). This allows many different potential uses including virtual key pads and other controls.

In this simple example, we will implement a touch sensitive button and a virtual led that lights up when the button is pressed (as shown in Figure 13).

Figure 13 – The touchscreen example

Code listing 4 shows the use of the stm32f7_discovery board board support package touch screen API (as always, the full project can be found on GitHub).

Code listing 4:

```
/*
 * main.c
 *
 * this is the main simple touch screen application example
 *
 * author:    Dr. Alex Shenfield
 * date:      05/10/2021
 * purpose:   55-604481 embedded computer networks : lab 102
 */

// include the basic headers and hal drivers
#include "stm32f7xx_hal.h"

// include the shu bsp libraries for the stm32f7 discovery board
#include "pinmappings.h"
#include "clock.h"

#include "stm32746g_discovery_lcd.h"
#include "stm32746g_discovery_ts.h"

// CODE

// this is the main method
int main()
{
  // we need to initialise the hal library and set up the SystemCoreClock
  // properly
  HAL_Init();
  init_sysclk_216MHz();

  // initialise the lcd
  BSP_LCD_Init();
  BSP_LCD_LayerDefaultInit(LTDC_ACTIVE_LAYER, SDRAM_DEVICE_ADDR);
  BSP_LCD_SelectLayer(LTDC_ACTIVE_LAYER);

  // set the background colour to black and clear the lcd
  BSP_LCD_SetBackColor(LCD_COLOR_BLACK);
  BSP_LCD_Clear(LCD_COLOR_BLACK);

  // draw a blue button
  BSP_LCD_SetTextColor(LCD_COLOR_BLUE);
  BSP_LCD_FillRect(50, 110, 145, 52);

  // write text on to it
  BSP_LCD_SetFont(&Font20);
  BSP_LCD_SetBackColor(LCD_COLOR_BLUE);
  BSP_LCD_SetTextColor(LCD_COLOR_GREEN);
  BSP_LCD_DisplayStringAt(60, 128, (uint8_t *)"Press me!", LEFT_MODE);

  // initialise a toggle state variable for our virtual led
  uint8_t led = 0;

  // draw our virtual led
  BSP_LCD_SetBackColor(LCD_COLOR_BLACK);
  BSP_LCD_SetTextColor(LCD_COLOR_YELLOW);
  BSP_LCD_DrawCircle(350, 136, 26);

  // initialise the touch screen state structure to store information about
  // the touches
  TS_StateTypeDef TS_State;
  BSP_TS_Init(480, 272);
```

```
  // main loop
  while(1)
  {
    // get the state of the touch screen
    BSP_TS_GetState(&TS_State);

    // if a touch is detected ...
    if(TS_State.touchDetected)
    {
      // work out if any of the touches are in the area of the button
      uint8_t idx;
      for(idx = 0; idx < TS_State.touchDetected; idx++)
      {
        uint16_t x = TS_State.touchX[idx];
        uint16_t y = TS_State.touchY[idx];
        if(x > 50 && x < 195 && y > 110 && y < 162)
        {
          led = !led;
        }
      }

      // update the "led"
      if(led)
      {
        // draw a filled "led"
        BSP_LCD_SetBackColor(LCD_COLOR_BLACK);
        BSP_LCD_SetTextColor(LCD_COLOR_YELLOW);
        BSP_LCD_FillCircle(350, 136, 26);
      }
      else
      {
        // clear the led
        BSP_LCD_SetTextColor(LCD_COLOR_BLACK);
        BSP_LCD_FillCircle(350, 136, 26);

        // draw an empty "led"
        BSP_LCD_SetBackColor(LCD_COLOR_BLACK);
        BSP_LCD_SetTextColor(LCD_COLOR_YELLOW);
        BSP_LCD_DrawCircle(350, 136, 26);
      }

      // dumb ass delay rather than debouncing because i'm lazy :)
      HAL_Delay(500);
    }
  }
}
```

Build and run the code provided and verify it works.

Appendix A – Extracts from the TMP36 data sheet

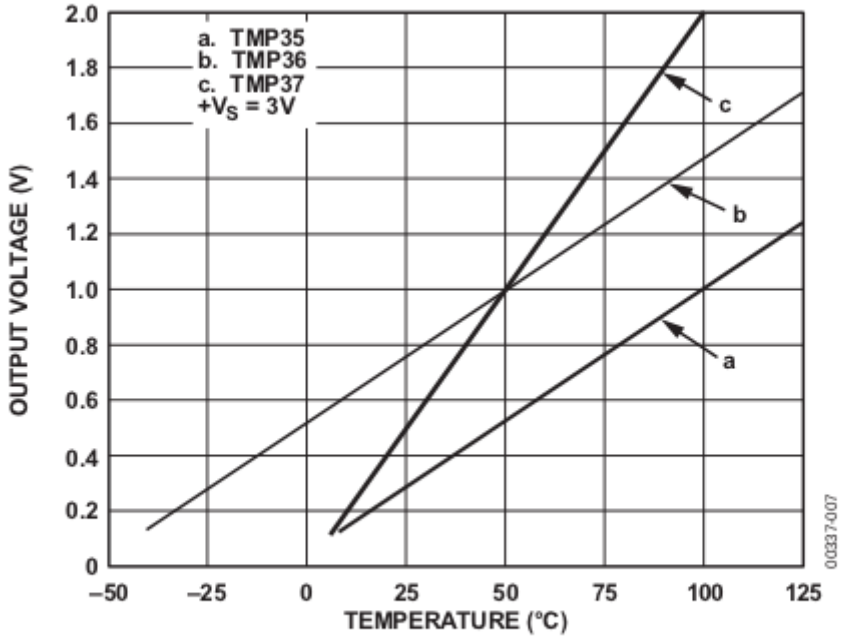
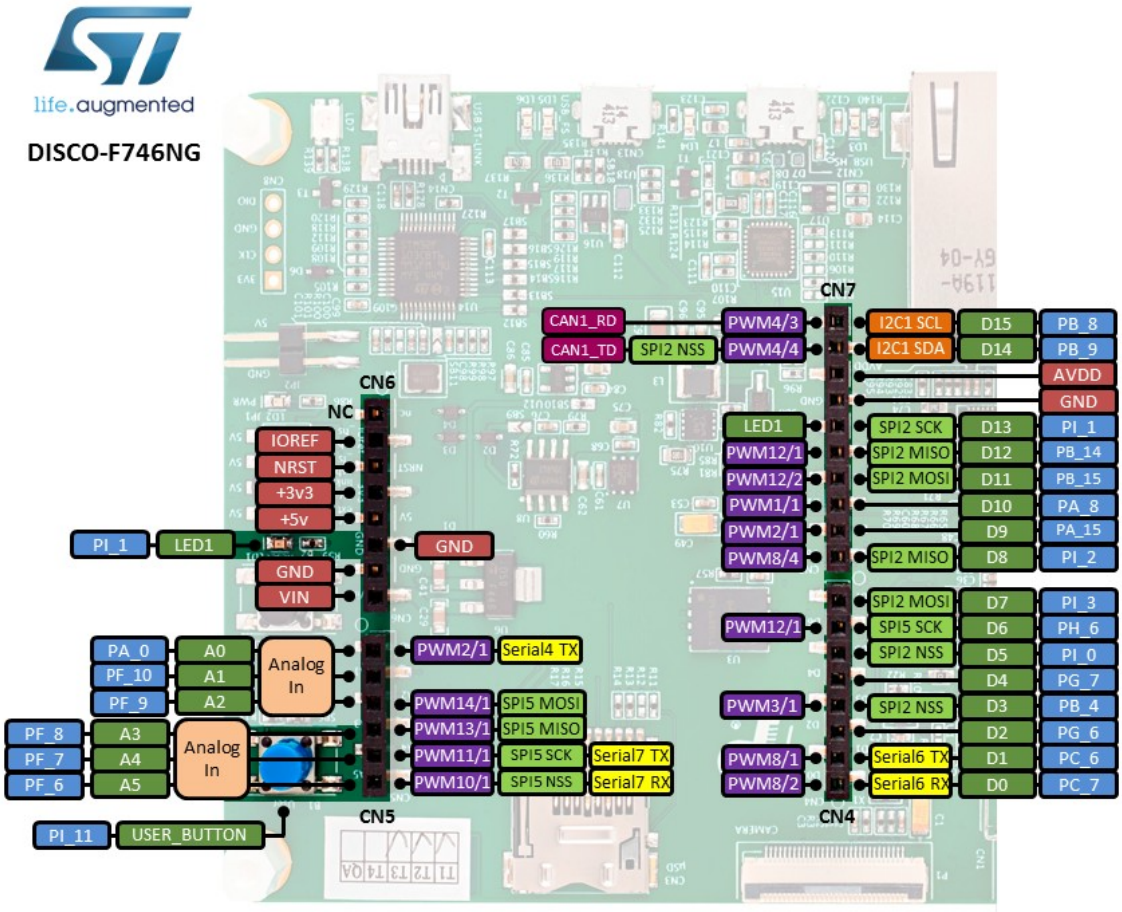


Figure 6. Output Voltage vs. Temperature

Table 4. TMP3x Output Characteristics

Sensor	Offset Voltage (V)	Output Voltage Scaling (mV/°C)	Output Voltage @ 25°C (mV)
TMP35	0	10	250
TMP36	0.5	10	750
TMP37	0	20	500

Appendix B – The STM32F7 discovery board schematic



STM32F7 discovery board pin outs