***Efficient Embedded Course***

**LAB 4**

**GENERAL PURPOSE I/O LAB EXERCISE:**

**BASIC USER INTERFACE**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc87592909)

[1.1 Lab overview 1](#_Toc87592910)

[2 Learning Objectives 1](#_Toc87592911)

[3 Requirements 1](#_Toc87592912)

[4 Switch Interfacing 2](#_Toc87592913)

[4.1 Hardware 2](#_Toc87592914)

[4.1.1 Connections 2](#_Toc87592915)

[4.2 Software 3](#_Toc87592916)

[4.2.1 Switch Interface Code 3](#_Toc87592917)

[4.2.2 RGB LED Interface Code 3](#_Toc87592918)

[4.2.3 Application Code 4](#_Toc87592919)

[4.3 LCD Interfacing 4](#_Toc87592920)

[4.3.1 Hardware 4](#_Toc87592921)

[4.3.2 Software 6](#_Toc87592922)

# Introduction

## Lab overview

**

Figure 1. 5-way switch  Figure 2. A two line by 24 character LCD module.

For this project you will add switches (see Figure 1) and a text LCD (see Figure 2) to create a simple user interface for the Nucleo-F401RE board. With it you can control your microcontroller and display information for the user.

# Learning Objectives

* Implement a simple C program to read the states of external switches
* Implement a simple C program to turn on/off the external LEDs.
* Implement a simple C program to write values to an LCD.

# Requirements

In this lab, we will be using the following hardware and software:

* **Keil µVision5 MDK IDE**
  + Please see the included Getting Started with Keil guide on how to download and install Keil.
* **STM32 Nucleo-L552ZE-Q**
  + For more information, click [here](https://www.st.com/en/evaluation-tools/nucleo-l552ze-q.html).
* **5-way switch**
  + Alternatively, this can be made using five buttons/switches as described in *Figure 3*.
* **LCD module**

# Switch Interfacing

## Hardware



Figure 3. Schematic diagram

Use five momentary switches SW1-SW5 or one multi-way switch to generate user input. You will use the MCU’s built-in pull-up resistors to ensure the signals are logic ones when the switches are open.

### Connections

Connect the switch signals to GPIO port signals on the MCU as shown in table below. This matches the pins used in the furnished code.

Table 1. Switch signals and connections

|  |  |  |  |
| --- | --- | --- | --- |
| Signal Name | Description | Direction | MCU |
| SWUp | Up | Input to MCU | PA\_1 |
| SWDn | Down | Input to MCU | PB\_0 |
| SWLt | Left | Input to MCU | PC\_1 |
| SWRt | Right | Input to MCU | PC\_0 |
| SWCr | Center | Input to MCU | PA\_4 |
| VSS | Ground |  |  |

Please see the Nucleo-L552ZE-Q User manual for the pinout of the Arduino-included Zio connectors for CN7, CN8, CN9 and CN10 using this link: https://www.st.com/resource/en/user\_manual/um2581-stm32l5-nucleo144-board-mb1361-stmicroelectronics.pdf

## Software

### Switch Interface Code

Let’s see how to use the GPIO pins to interface a Cortex-M4 MCU to a 5-way switch. First, we define a function to initialize all the switches with pull-up resistors.

void switches\_init(void) {

gpio\_set\_mode(P\_SW\_UP, PullUp);

gpio\_set\_mode(P\_SW\_DN, PullUp);

gpio\_set\_mode(P\_SW\_LT, PullUp);

gpio\_set\_mode(P\_SW\_RT, PullUp);

gpio\_set\_mode(P\_SW\_CR, PullUp);

}

Next we define a function to read a switch. Note that a switch returns a value of 0 when pressed (active-LOW).

int switch\_get(Pin pin) {

return !gpio\_get(pin);

}

Now we can write code to detect when switches are pressed. We will light up different LEDs based on which switches are pressed.

### RGB LED Interface Code

We also initialize the RGB LEDs on the MCU board. The LED positions are defined in platform.h.

void leds\_init(void) {

// Set 3 led pins to outputs.

gpio\_set\_mode(P\_LED\_R, Output);

gpio\_set\_mode(P\_LED\_G, Output);

gpio\_set\_mode(P\_LED\_B, Output);

leds\_set(0, 0, 0);

}

The leds\_set function is straightforward; keep in mind the LEDs are active-LOW however.

void leds\_set(int red\_on, int green\_on, int blue\_on) {

// LEDs are active-LOW

gpio\_set(P\_LED\_R, !red\_on);

gpio\_set(P\_LED\_G, !green\_on);

gpio\_set(P\_LED\_B, !blue\_on);

}

### Application Code

In the main function, we initialize the switches and LEDs and then call the print\_switches and light\_leds function.

switches\_init();

leds\_init();

lcd\_init();

lcd\_print("Hello World!");

while (1) {

print\_switches();

light\_leds();

}

Within light\_leds, we read the switches switch\_get function, and pass the result to leds\_set to specify whether each LED should be lit.

void light\_leds() {

leds\_set(switch\_get(P\_SW\_UP),

switch\_get(P\_SW\_CR),

switch\_get(P\_SW\_DN));

}

## LCD Interfacing

### Hardware

#### LCD Controller Concepts

***A close-up of a circuit board

Description automatically generated with medium confidence***

Figure 5. Back of module showing controller and column driver ICs.

The character LCD module shown in Figure 2 consists of an LCD glass panel and several digital logic ICs on the back of the module, shown in Figure 5. The LCD is controlled by an HD44780 or compatible LCD controller IC (in this case, an M6222B-01) and several subordinate LCD driver ICs (LH5006A).



Figure 6. Block diagram of character LCD module and interface with MCU.

Figure 6 presents a block diagram of the LCD control system and the interface to a microcontroller. The interface consists of three control lines (Enable (E), Read/~Write (R/~W)), and Register Select (RS), and four or eight data lines (DB4-7 or DB8-7). The falling edge of the E signal triggers an operation based on the state of RS and R/~W, as shown in Table 2.

Table 2. HD44780 LCD Controller Operations

|  |  |  |
| --- | --- | --- |
| RS | R/~W | Operation |
| 0 | **0** | Write instruction |
| 0 | **1** | Read busy flag and address counter |
| 1 | **0** | Write data |
| 1 | **1** | Read data |

The following table shows the typical signal assignments for the LCD interface on a 14-pin connector.

Table 3. HD44780 LCD Controller Interface

|  |  |  |
| --- | --- | --- |
| Pin | Signal | Description |
| 1 | VSS | Ground |
| 2 | VDD | Power supply |
| 3 | VO | LCD contrast adjustment voltage |
| 4 | RS | Register Select |
| 5 | R/W | Read |
| 6 | E | Enable (Clock signal) |
| 7 | DB0 | Data bus (LSB) |
| 8 | DB1 | Data bus |
| 9 | DB2 | Data bus |
| 10 | DB3 | Data bus |
| 11 | DB4 | Data bus |
| 12 | DB5 | Data bus |
| 13 | DB6 | Data bus |
| 14 | DB7 | Data bus (MSB) |

Some LCD controllers have LED backlighting. These may use a 16-pin connector, with pins 15 and 16 connected to the anode and cathode of the LED.

#### Connections

Allocate the switch and LCD controller signals to GPIO port signals on the MCU. The table below shows the allocations used for the furnished code.

Table 4. LCD signals and connections

|  |  |  |  |
| --- | --- | --- | --- |
| Signal Name | Description | Direction | MCU |
| E | LCDEnable | Output from MCU | PB\_10 |
| R/W | Read /Write | Output from MCU | PA\_8 |
| RS | RegisterSelect | Output from MCU | PA\_9 |
| DB4 | Data bus 4 | I/O to/from MCU | PB\_4 |
| DB5 | Data bus 5 | I/O to/from MCU | PB\_5 |
| DB6 | Data bus 6 | I/O to/from MCU | PB\_3 |
| DB7 | Data bus 7 | I/O to/from MCU | PA\_10 |
| VDD | Supply Voltage | Input to LCD Panel |  |
| VSS | Ground | Input to LCD Panel |  |

### Software

#### LCD Interface Code

Let’s see how to use the GPIO pins to interface an MCU to an LCD module.

The MCU always drives the three control lines, so they need to be outputs with normal drive capabilities. The data lines are typically written, but may also be read for status information, so they need to be inputs or outputs depending on the communication operation with the LCD. We will initialize them as outputs but switch them to inputs when needed. As inputs, they do not need pull-up or pull-down resistors. As outputs, they only need normal drive capabilities. Let’s use the 4-bit data interface to save pins and wiring effort.

* We use the GPIO drivers to set the data pins as inputs or outputs.

static void set\_data\_dir(PinMode mode) {

uint8\_t i;

for (i = 0; i < 4; i++) {

gpio\_set\_mode((Pin)(P\_LCD\_DATA[i]), mode);

}

}

* To write data on the 4-bit data bus, we assert the control lines as specified in the LCD controller data manual.

void lcd\_write\_4bit(uint8\_t c) {

int i=0;

gpio\_set(P\_LCD\_RW, 0);

gpio\_set(P\_LCD\_E, 1);

for(i=0; i<4; i++) {

gpio\_set(P\_LCD\_DATA[i], (c & (0x1<<i))>>i);

}

delay\_us(1);

gpio\_set(P\_LCD\_E, 0);

delay\_us(1);

}

* To read from the LCD, we need to switch the data bus direction to input, read 4 bits of data (the upper nibble), read in 4 more bits of data (the lower nibble), form these nibbles into a byte, and then switch the data bus direction back to output.

uint8\_t lcd\_read\_status(void) {

uint8\_t status;

int i;

set\_data\_dir(Input);

gpio\_set(P\_LCD\_RS, 0);

gpio\_set(P\_LCD\_RW, 1);

delay\_us(1);

gpio\_set(P\_LCD\_E, 1);

delay\_us(1);

for(i=0; i<4; i++) {

status |= gpio\_get(P\_LCD\_DATA[i]) << (4-i);

}

gpio\_set(P\_LCD\_E, 0);

delay\_us(1);

gpio\_set(P\_LCD\_E, 1);

delay\_us(1);

for(i=0; i<4; i++) {

status |= gpio\_get(P\_LCD\_DATA[i]) << (4-i);

}

gpio\_set(P\_LCD\_E, 0);

set\_data\_dir(Output);

return status;

}

* We can use these pieces to initialize the HD44780 LCD controller, as directed in the datasheet.

void lcd\_init(void) {

// Set all LCD pins as outputs

gpio\_set\_mode(P\_LCD\_RS, Output);

gpio\_set\_mode(P\_LCD\_RW, Output);

gpio\_set\_mode(P\_LCD\_E, Output);

set\_data\_dir(Output);

// Have to wait 100ms or more after Power-on

delay\_ms(100);

// Run LCD initialization sequence

// Set the device to change mode from 8-bit to 4-bit

lcd\_write\_cmd(0x30);

delay\_ms(5);

lcd\_write\_cmd(0x30);

delay\_us(100);

lcd\_write\_cmd(0x30);

delay\_us(100);

lcd\_write\_cmd(0x20);

delay\_us(100);

// The device is in 8-bit mode when powered up

// At this point it switches to 4-bit mode

// Configure operating mode (number of rows/lines)

lcd\_write\_cmd(0x28);

delay\_us(60);

// Set display off, under-line off and block cursor off

lcd\_write\_cmd(0x08);

delay\_us(60);

// Clear display

lcd\_write\_cmd(0x01);

delay\_ms(4);

// Set the direction of address counter to increment

// and no display shift on data read/write

lcd\_write\_cmd(0x06);

delay\_us(60);

// Set display on, under-line off and block cursor off

lcd\_write\_cmd(0x0C);

delay\_us(60);

lcd\_set\_cursor(0,0);

lcd\_set\_cursor\_visible(0);

}

#### Hello, World

Let’s use these modules to create a program which first displays “Hello World!” on the LCD and displays different strings depending on which switch the user has pressed. Within main, we call lcd\_init to initialize the GPIO ports which the LCD controller is connected to, and then initialize the LCD controller itself. We then clear the LCD and display the message.

switches\_init();

leds\_init();

lcd\_init();

lcd\_print("Hello World!");

#### Switch Reading and Reporting

Finally, let’s write a function which indicates on the LCD which switch is pressed. First, we set the cursor to the beginning of the area for our message. After reading the switch code, we test until finding the first pressed switch, at which point we print out the name of the switch. If no switches are pressed, we erase anything that was printed there previously.

void print\_switches() {

lcd\_set\_cursor(0, 1);

if (switch\_get(P\_SW\_UP)) {

lcd\_print(" Lorem ");

} else if (switch\_get(P\_SW\_CR)) {

lcd\_print(" ipsum ");

} else if (switch\_get(P\_SW\_DN)) {

lcd\_print(" dolor ");

} else if (switch\_get(P\_SW\_LT)) {

lcd\_print(" sit ");

} else if (switch\_get(P\_SW\_RT)) {

lcd\_print(" amet ");

} else {

lcd\_print(" ");

}

}