**General Purpose I/O Lab Exercise:**

**Basic User Interface**

**Issue 1.0**

# Overview

|  |  |
| --- | --- |
| C:\Users\Alex\Documents\Teaching\Book Writin'\ARM Cortex M0Plus\Content\GPIO\Figures\5-way Switch crop.jpg  Figure . 5-way switch | *C:\Users\Alex\Documents\Teaching\Book Writin'\ARM M0Plus\Content\GPIO\Figures\LCD Photos\LCD Front.JPG*  Figure . 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 Freedom KL25Z board. With it you can control your microcontroller and display information for the user.

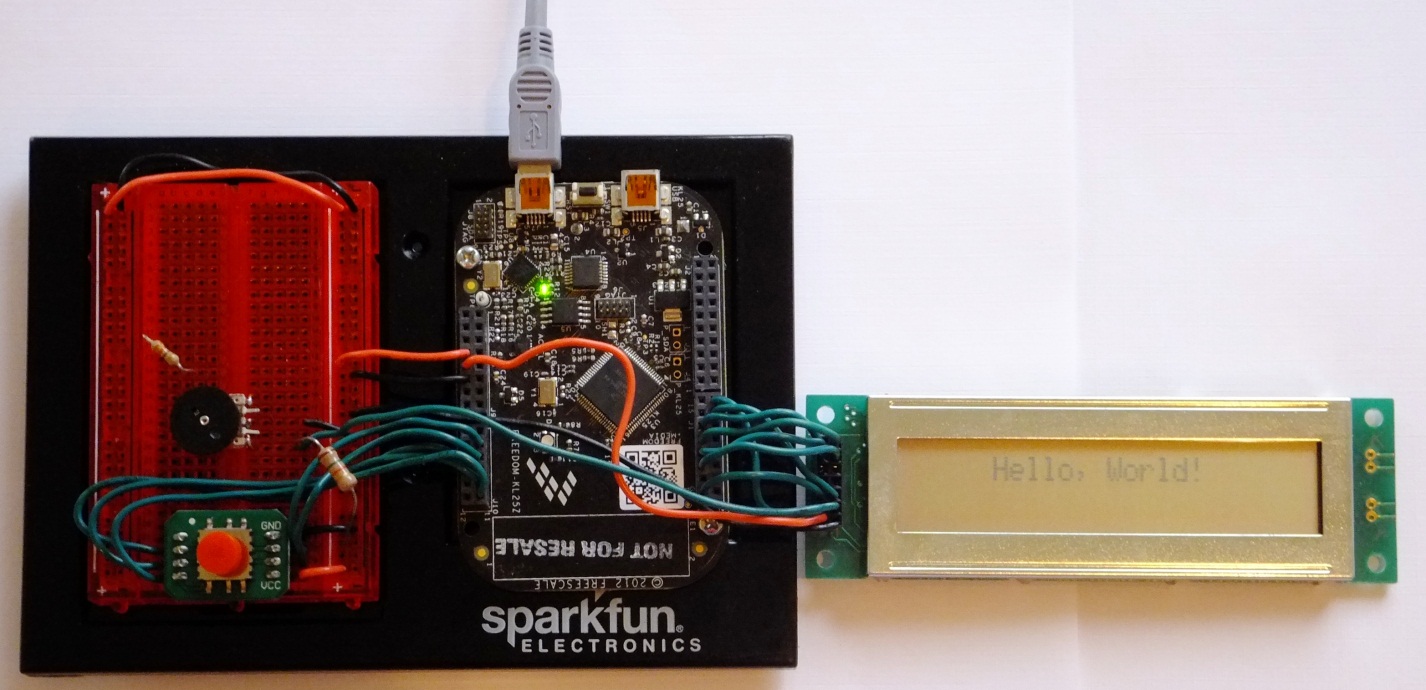


Figure . Freedom KL25Z Board connected with switch and LCD.

# Switch Interfacing

## Hardware



Figure . 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.

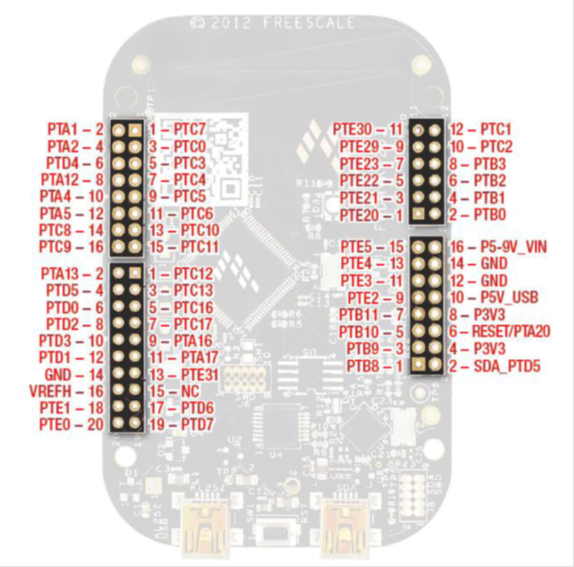
### Connections

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 . Switch signals and connections

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Signal Name | Description | Direction | MCU | | Freedom KL25Z Board | |
| **Port** | **Bit** | **Connector** | **Pin** |
| SWUp | Up | Input to MCU | E | 21 | J10 | 3 |
| SWDn | Down | Input to MCU | E | 29 | J10 | 9 |
| SWLt | Left | Input to MCU | E | 30 | J10 | 11 |
| SWRt | Right | Input to MCU | E | 23 | J10 | 7 |
| SWCr | Center | Input to MCU | E | 22 | J10 | 5 |
| VSS | Ground |  |  |  | J9 | 14 |

Warning: the connectors on the KL25Z are oriented such that J1/J2 and J10/J11 are rotated 180° from each other, as shown in Figure 5. Verify where pin 1 is for each connector.



**J1**

**J2**

**J10**

**J9**

Figure . Freedom KL25Z I/O connectors with pin 1 marked on each with an arrow.

## Software

### Switch Interface Code

Let’s see how to use the GPIO pins to interface a Cortex M0+ MCU to a 5-way switch using port E. First, we define the bit positions of all of the switches matching the connections provided above .

// All switches are on port E

#define SW\_UP\_POS (21)

#define SW\_DN\_POS (29)

#define SW\_LT\_POS (30)

#define SW\_RT\_POS (23)

#define SW\_CR\_POS (22)

Next we define a macro to read the switches. Note that a switch returns a value of 0 when pressed (active low).

// Macro to read switches returns state of switches, active low

#define READ\_SWITCHES (PTE->PDIR)

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

First, we initialize the GPIO ports which are connected to the switches.

void init\_5way\_switch(void) {

SIM->SCGC5 |= SIM\_SCGC5\_PORTE\_MASK; /\* enable clock for port E \*/

/\* Select GPIO and enable pull-up resistors for pins connected to switches \*/

PORTE->PCR[SW\_UP\_POS] |= PORT\_PCR\_MUX(1) | PORT\_PCR\_PS\_MASK | PORT\_PCR\_PE\_MASK;

PORTE->PCR[SW\_DN\_POS] |= PORT\_PCR\_MUX(1) | PORT\_PCR\_PS\_MASK | PORT\_PCR\_PE\_MASK;

PORTE->PCR[SW\_LT\_POS] |= PORT\_PCR\_MUX(1) | PORT\_PCR\_PS\_MASK | PORT\_PCR\_PE\_MASK;

PORTE->PCR[SW\_RT\_POS] |= PORT\_PCR\_MUX(1) | PORT\_PCR\_PS\_MASK | PORT\_PCR\_PE\_MASK;

PORTE->PCR[SW\_CR\_POS] |= PORT\_PCR\_MUX(1) | PORT\_PCR\_PS\_MASK | PORT\_PCR\_PE\_MASK;

/\* Set port C bits 0-3, 7 to inputs \*/

PTE->PDDR &= ~( GPIO\_PDDR\_PDD(SW\_UP\_POS) |

GPIO\_PDDR\_PDD(SW\_DN\_POS) |

GPIO\_PDDR\_PDD(SW\_LT\_POS) |

GPIO\_PDDR\_PDD(SW\_RT\_POS) |

GPIO\_PDDR\_PDD(SW\_CR\_POS) );

}

### RGB LED Interface Code

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

void init\_RGB\_LEDs(void) {

// Enable clock to ports B and D

SIM->SCGC5 |= SIM\_SCGC5\_PORTB\_MASK | SIM\_SCGC5\_PORTD\_MASK;;

// Make 3 pins GPIO

PORTB->PCR[RED\_LED\_POS] &= ~PORT\_PCR\_MUX\_MASK;

PORTB->PCR[RED\_LED\_POS] |= PORT\_PCR\_MUX(1);

PORTB->PCR[GREEN\_LED\_POS] &= ~PORT\_PCR\_MUX\_MASK;

PORTB->PCR[GREEN\_LED\_POS] |= PORT\_PCR\_MUX(1);

PORTD->PCR[BLUE\_LED\_POS] &= ~PORT\_PCR\_MUX\_MASK;

PORTD->PCR[BLUE\_LED\_POS] |= PORT\_PCR\_MUX(1);

// Set ports to outputs

PTB->PDDR |= MASK(RED\_LED\_POS) | MASK(GREEN\_LED\_POS);

PTD->PDDR |= MASK(BLUE\_LED\_POS);

}

The control\_RGB\_LEDs function is straightforward, and uses the port set and port clear registers.

void control\_RGB\_LEDs(unsigned int red\_on, unsigned int green\_on, unsigned int blue\_on) {

if (red\_on) {

PTB->PCOR = MASK(RED\_LED\_POS);

} else {

PTB->PSOR = MASK(RED\_LED\_POS);

}

if (green\_on) {

PTB->PCOR = MASK(GREEN\_LED\_POS);

} else {

PTB->PSOR = MASK(GREEN\_LED\_POS);

}

if (blue\_on) {

PTD->PCOR = MASK(BLUE\_LED\_POS);

} else {

PTD->PSOR = MASK(BLUE\_LED\_POS);

}

}

### Application Code

In the main function we initialize the switches and LEDs and then call the test\_switches function.

init\_5way\_switch();

init\_RGB\_LEDs();

test\_switches();

Within test\_switches, we use an infinite loop to read the switches using the switch macro. We invert the result of the macro in order to do our logic with active-high rather than active-low signals. This makes the code easier to understand. We then call control\_RGB\_LEDs with three arguments to specify whether each LED should be lit.

void test\_switches(void) {

unsigned switch\_code;

while (1) {

switch\_code = ~READ\_SWITCHES;

control\_RGB\_LEDs( (switch\_code & MASK(SW\_UP\_POS)),

(switch\_code & MASK(SW\_RT\_POS)),

(switch\_code & MASK(SW\_LT\_POS)) );

}

}

# LCD Interfacing

## Hardware

### LCD Controller Concepts

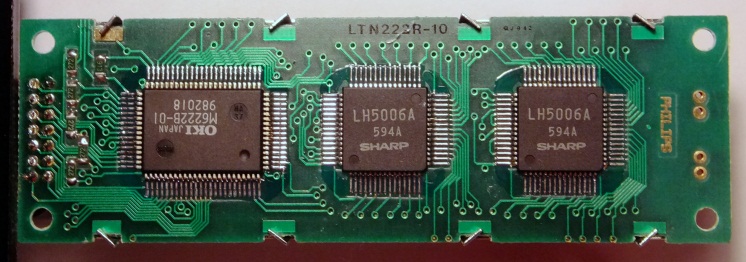
******

Figure . 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 inFigure 6. The LCD is controlled by an HD44780 or compatible LCD controller IC (in this case, an M6222B-01) and several slave LCD driver ICs (LH5006A).



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

Figure 7 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 upon the state of RS and R/~W, as shown in Table 2.

Table . 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 . 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

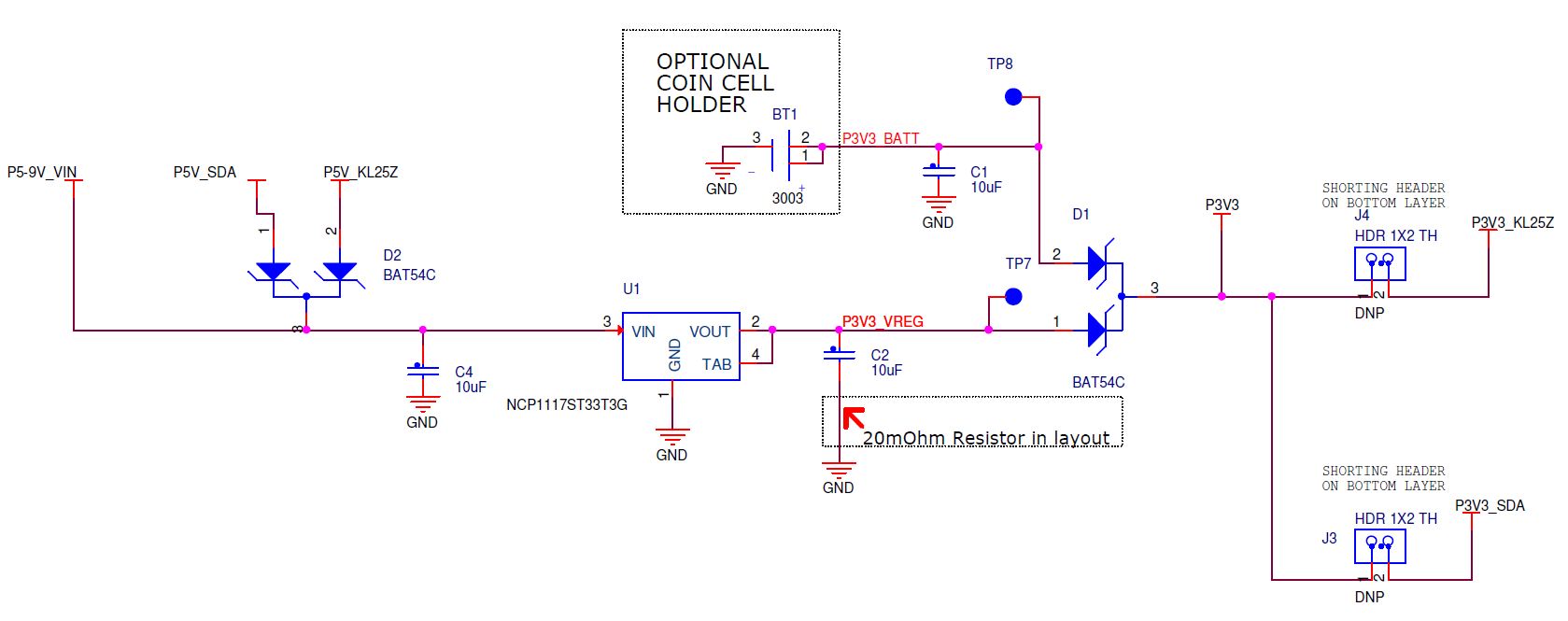
Allocate the switch and LCD controller signals to GPIO port signals on the MCU. The table below shows the allocations used for the furnished code.

Table . LCD signals and connections

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Signal Name | Description | Direction | MCU | | Freedom KL25Z Board | |
| **Port** | **Bit** | **Connector** | **Pin** |
| E | LCDEnable | Output from MCU | C | 7 | J1 | 6 |
| R/W | Read /Write | Output from MCU | C | 8 | J1 | 14 |
| RS | RegisterSelect | Output from MCU | C | 9 | J1 | 16 |
| DB4 | Data bus 4 | I/O to/from MCU | C | 3 | J1 | 11 |
| DB5 | Data bus 5 | I/O to/from MCU | C | 4 | J1 | 12 |
| DB6 | Data bus 6 | I/O to/from MCU | C | 5 | J1 | 13 |
| DB7 | Data bus 7 | I/O to/from MCU | C | 6 | J1 | 14 |
| VDD | Supply Voltage | Input to LCD Panel |  |  | J9 | 10 |
| VSS | Ground | Input to LCD Panel |  |  | J9 | 14 |
| VO | Contrast Adjustment | Input to LCD Panel |  |  |  |  |

Warning: the connectors on the KL25Z are oriented such that J1/J2 and J10/J11 are rotated 180° from each other, as shown in Figure 5. Again, be sure to verify where pin 1 is for each connector.

### LCD Module Operating Voltage



**~5.1V**

**~4.95V**

**3.3V**

Figure . Power supply for Freedom KL25Z board.

The power supply of the Freedom KL25Z is shown in Figure 8. Voltage measurements are added to show levels when the system is connected to a powered USB through the SDA USB connector. Note the use of diodes to protect the circuits if multiple power sources are connected simultaneously. The regulated 3.3 V supply is too low to operate some LCDs – examine the LCD’s datasheet to verify the minimum supply voltage.

* For a 3.3 V LCD, connect the LCD VDD supply line to the KL25Z’s nominal 3.3V supply rail (P3V3 pins 4 and 8 of header J9).
* For a 5 V LCD, connect the LCD VDD to P5V\_USB (P5V\_USB, pin 10 of header J9) which is approximately 5 V.

## Software

### LCD Interface Code

Let’s see how to use the GPIO pins to interface a Cortex M0+ MCU to an LCD module using port C.

* We begin by assigning the four data bus signals to Port C bits 3 through 6 and the three control signals to Port C bits 7 through 9. We use #define macros to simplify code development and maintenance. The data starts at bit 3, while the control lines start at bit 7.

#define PIN\_DATA\_PORT PORTC

#define PIN\_DATA\_PT PTC

#define PIN\_DATA\_SHIFT ( 3 )

#define PIN\_E\_PORT PORTC

#define PIN\_E\_PT PTC

#define PIN\_E\_SHIFT ( 7 )

#define PIN\_E ( 1 << PIN\_E\_SHIFT)

#define PIN\_RW\_PORT PORTC

#define PIN\_RW\_PT PTC

#define PIN\_RW\_SHIFT ( 8 )

#define PIN\_RW ( 1 << PIN\_RW\_SHIFT)

#define PIN\_RS\_PORT PORTC

#define PIN\_RS\_PT PTC

#define PIN\_RS\_SHIFT ( 9 )

#define PIN\_RS ( 1 << PIN\_RS\_SHIFT)

#define PINS\_DATA (0x0F << PIN\_DATA\_SHIFT)

* We enable the Port C clock signal.

#define ENABLE\_LCD\_PORT\_CLOCKS SIM->SCGC5 |= SIM\_SCGC5\_PORTC\_MASK;

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 create three macros to set port pin directions appropriately, using the pin definition macros created previously.

#define SET\_LCD\_ALL\_DIR\_OUT { \

PIN\_DATA\_PT->PDDR = PIN\_DATA\_PT->PDDR | PINS\_DATA; \

PIN\_E\_PT->PDDR = PIN\_E\_PT->PDDR | PIN\_E; \

PIN\_RW\_PT->PDDR = PIN\_RW\_PT->PDDR | PIN\_RW; \

PIN\_RS\_PT->PDDR = PIN\_RS\_PT->PDDR | PIN\_RS; }

#define SET\_LCD\_DATA\_DIR\_IN PIN\_DATA\_PT->PDDR = PIN\_DATA\_PT->PDDR & ~PINS\_DATA;

#define SET\_LCD\_DATA\_DIR\_OUT PIN\_DATA\_PT->PDDR = PIN\_DATA\_PT->PDDR | PINS\_DATA;

* We create macros to access the data bus, shifting based on the offset.

#define SET\_LCD\_DATA\_OUT(x) PIN\_DATA\_PT->PDOR = (PIN\_DATA\_PT->PDOR & ~PINS\_DATA) | ((x) << PIN\_DATA\_SHIFT);

#define GET\_LCD\_DATA\_IN (((PIN\_DATA\_PT->PDIR & PINS\_DATA) >> PIN\_DATA\_SHIFT) & 0x0F)

* We create macros to manipulate the control lines.

#define SET\_LCD\_E(x) if (x) {PIN\_E\_PT->PSOR = PIN\_E;} else {PIN\_E\_PT->PCOR = PIN\_E;}

#define SET\_LCD\_RW(x) if (x) {PIN\_RW\_PT->PSOR = PIN\_RW;} else {PIN\_RW\_PT->PCOR = PIN\_RW;}

#define SET\_LCD\_RS(x) if (x) {PIN\_RS\_PT->PSOR = PIN\_RS;} else {PIN\_RS\_PT->PCOR = PIN\_RS;}

* We can now use these pieces to initialize the LCD port pins.

void lcd\_init\_port(void) {

/\* Enable clocks for peripherals \*/

ENABLE\_LCD\_PORT\_CLOCKS

/\* Set Pin Mux to GPIO \*/

PIN\_DATA\_PORT->PCR[PIN\_DATA\_SHIFT] = PORT\_PCR\_MUX(1);

PIN\_DATA\_PORT->PCR[PIN\_DATA\_SHIFT+1] = PORT\_PCR\_MUX(1);

PIN\_DATA\_PORT->PCR[PIN\_DATA\_SHIFT+2] = PORT\_PCR\_MUX(1);

PIN\_DATA\_PORT->PCR[PIN\_DATA\_SHIFT+3] = PORT\_PCR\_MUX(1);

PIN\_E\_PORT->PCR[PIN\_E\_SHIFT] = PORT\_PCR\_MUX(1);

PIN\_RW\_PORT->PCR[PIN\_RW\_SHIFT] = PORT\_PCR\_MUX(1);

PIN\_RS\_PORT->PCR[PIN\_RS\_SHIFT] = PORT\_PCR\_MUX(1);

}

* 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)

{

SET\_LCD\_RW(0)

SET\_LCD\_E(1)

SET\_LCD\_DATA\_OUT(c&0x0F)

Delay(1);

SET\_LCD\_E(0)

Delay(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;

SET\_LCD\_DATA\_DIR\_IN

SET\_LCD\_RS(0)

SET\_LCD\_RW(1)

Delay(1);

SET\_LCD\_E(1)

Delay(1);

status = GET\_LCD\_DATA\_IN << 4;

SET\_LCD\_E(0)

Delay(1);

SET\_LCD\_E(1)

Delay(1);

status |= GET\_LCD\_DATA\_IN;

SET\_LCD\_E(0)

SET\_LCD\_DATA\_DIR\_OUT

return(status);

}

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

void Init\_LCD(void)

{

/\* initialize port(s) for LCD \*/

lcd\_init\_port();

/\* Set all pins for LCD as outputs \*/

SET\_LCD\_ALL\_DIR\_OUT

Delay(100);

SET\_LCD\_RS(0)

lcd\_write\_4bit(0x3);

Delay(100);

lcd\_write\_4bit(0x3);

Delay(10);

lcd\_write\_4bit(0x3);

lcd\_write\_4bit(0x2);

lcd\_write\_cmd(0x28);

lcd\_write\_cmd(0x0C);

lcd\_write\_cmd(0x06);

lcd\_write\_cmd(0x80);

}

### Hello, World

Let’s use these modules to create a program which first displays “Hello World!” on the LCD and counts up. 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, set the cursor and display the message.

lcd\_init();

lcd\_clear();

set\_cursor(0,0);

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 Test\_Switches\_And\_LCD(void) {

unsigned switch\_code;

while (1) {

Set\_Cursor(9,1);

switch\_code = ~READ\_SWITCHES;

if (switch\_code & MASK(SW\_UP\_POS)) {

Print\_LCD(" Up ");

} else if (switch\_code & MASK(SW\_DN\_POS)) {

Print\_LCD(" Down");

} else if (switch\_code & MASK(SW\_LT\_POS)) {

Print\_LCD(" Left");

} else if (switch\_code & MASK(SW\_RT\_POS)) {

Print\_LCD(" Right");

} else if (switch\_code & MASK(SW\_CR\_POS)) {

Print\_LCD("Center");

} else {

Print\_LCD(" ");

}

}

}