General Purpose I/O Lab Exercise:   
Basic User Interface

# Overview

Figure 1.NHD-0208BZ LCD display

For this project you will add a text LCD (see Figure 1) to create a simple user interface. With it you can display information for the user.

# LCD Interfacing

## Hardware

### LCD Controller Concepts

The character LCD module shown in 1consists of an LCD glass panel and several digital logic ICs on the back of the module, and a backlight. The LCD is controlled by a ST7066U or compatible LCD controller IC and several slave LCD driver ICs. It has 2 lines and each with 8 characters. The following figure shows a more capable LCD module with 2 lines × 8 characters.



Figure 3. Block diagram of character LCD module and interface with MCU.

Figure 3 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 1.ST7066U 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 2.ST7066U 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

There are usually two ways to control the LCD: 4-bit and 8-bit. 8-bit control enables us to display more types of characters (e.g. Japanese katakana). However, 4-bit will suffice in most cases and we are using the 4-bit control for this lab. Therefore we can leave DB0-DB3 along this time.

Allocate controller signals to GPIO port signals on the board. The table below shows the allocations used for the furnished code.

Table 3. LCD signals and connections

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Signal Name | Description | Direction | Board | |
| **Port** | **Bit** |
| E | LCDEnable | Output from MCU | 1 | 10 |
| R/W | Read /Write | Output from MCU | 1 | 11 |
| RS | RegisterSelect | Output from MCU | 2 | 10 |
| DB4 | Data bus 4 | I/O to/from MCU | 2 | 6 |
| DB5 | Data bus 5 | I/O to/from MCU | 2 | 7 |
| DB6 | Data bus 6 | I/O to/from MCU | 2 | 8 |
| DB7 | Data bus 7 | I/O to/from MCU | 2 | 9 |
| VDD | Supply Voltage | Input to LCD Panel | 3.3V(3V) | |
| VSS | Ground | Input to LCD Panel | Ground | |
| VO | Contrast Adjustment | Input to LCD Panel | 0.1v | |

## 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 D.

* 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 LPC\_GPIO2

#define PIN\_DATA\_PT LPC\_GPIO2

#define PIN\_DATA\_SHIFT ( 6 )

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

#define PIN\_DATA\_PORT\_4 LPC\_GPIO2

#define PIN\_DATA\_PT\_4 6

#define PIN\_DATA\_SHIFT\_4 ( PIN\_DATA\_PT\_4 )

#define PIN\_DATA\_4 ( 1 << PIN\_DATA\_SHIFT\_4)

#define PIN\_DATA\_4\_01 (01 << (PIN\_DATA\_SHIFT\_4\*2))

#define PIN\_DATA\_4\_11 (11 << (PIN\_DATA\_SHIFT\_4\*2))

#define PIN\_DATA\_PORT\_5 LPC\_GPIO2

#define PIN\_DATA\_PT\_5 7

#define PIN\_DATA\_SHIFT\_5 ( PIN\_DATA\_PT\_5 )

#define PIN\_DATA\_5 ( 1 << PIN\_DATA\_SHIFT\_5)

#define PIN\_DATA\_5\_01 (01 << (PIN\_DATA\_SHIFT\_5\*2))

#define PIN\_DATA\_5\_11 (11 << (PIN\_DATA\_SHIFT\_5\*2))

#define PIN\_DATA\_PORT\_6 LPC\_GPIO2

#define PIN\_DATA\_PT\_6 8

#define PIN\_DATA\_SHIFT\_6 ( PIN\_DATA\_PT\_6 )

#define PIN\_DATA\_6 ( 1 << PIN\_DATA\_SHIFT\_6)

#define PIN\_DATA\_6\_01 (01 << (PIN\_DATA\_SHIFT\_6\*2))

#define PIN\_DATA\_6\_11 (11 << (PIN\_DATA\_SHIFT\_6\*2))

#define PIN\_DATA\_PORT\_7 LPC\_GPIO2

#define PIN\_DATA\_PT\_7 9

#define PIN\_DATA\_SHIFT\_7 ( PIN\_DATA\_PT\_7 )

#define PIN\_DATA\_7 ( 1 << PIN\_DATA\_SHIFT\_7)

#define PIN\_DATA\_7\_01 (01 << (PIN\_DATA\_SHIFT\_7\*2))

#define PIN\_DATA\_7\_11 (11 << (PIN\_DATA\_SHIFT\_7\*2))

#define PIN\_E\_PORT LPC\_GPIO1

#define PIN\_E\_PT 10

#define PIN\_E\_SHIFT ( PIN\_E\_PT )

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

#define PIN\_E\_01 (01 << (PIN\_E\_SHIFT\*2))

#define PIN\_RW\_PORT LPC\_GPIO1

#define PIN\_RW\_PT 11

#define PIN\_RW\_SHIFT ( PIN\_RW\_PT )

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

#define PIN\_RW\_01 (01 << (PIN\_RW\_SHIFT\*2))

#define PIN\_RS\_PORT LPC\_GPIO2

#define PIN\_RS\_PT 10

#define PIN\_RS\_SHIFT ( PIN\_RS\_PT )

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

#define PIN\_RS\_01 (01 << (PIN\_RS\_SHIFT\*2))

PT and PORT are actually the same thing but is maintained separately for compatibility (other MCU may have different configuration for the registers accessing mode). Also, if the DBs are assigned to non-consecutive pins, than the individual definition for each pin will be useful.

* We enable the Port D clock signal.

#define ENABLE\_LCD\_PORT\_CLOCKS LPC\_SYSCON->SYSAHBCLKCTRL |= (1<<6);

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. Be careful with these bits operations, could be easily wrong.

#define SET\_LCD\_E(x) if (x) {PIN\_E\_PORT->DATA |= PIN\_E;} else {PIN\_E\_PORT->DATA&=~PIN\_E;}

#define SET\_LCD\_RW(x) if (x) {PIN\_RW\_PORT->DATA |= PIN\_RW;} else {PIN\_RW\_PORT->DATA&=~PIN\_RW;}

#define SET\_LCD\_RS(x) if (x) {PIN\_RS\_PORT->DATA |= PIN\_RS;} else {PIN\_RS\_PORT->DATA&=~PIN\_RS;} We create macros to access the data bus, shifting based on the offset.

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

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

* We create macros to manipulate the control lines.

#define SET\_LCD\_E(x) if (x) {PIN\_E\_PORT->DATA |= PIN\_E;} else {PIN\_E\_PORT->DATA&=~PIN\_E;}

#define SET\_LCD\_RW(x) if (x) {PIN\_RW\_PORT->DATA |= PIN\_RW;} else {PIN\_RW\_PORT->DATA&=~PIN\_RW;}

#define SET\_LCD\_RS(x) if (x) {PIN\_RS\_PORT->DATA |= PIN\_RS;} else {PIN\_RS\_PORT->DATA&=~PIN\_RS;} We can now use these pieces to initialize the LCD port pins.

* 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.

voidInit\_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 “Hi,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("Hi, World");