CAB202 Topic 11 - LCD

Luis Mejias & Lawrence Buckingham, Queensland University of Technology.

Contents

Roadmap

Previously:

7. AVR ATMega328P Introduction to Microcontrollers; Digital Input/Output
8. Serial Communication - communicating with another computer/microcontroller
9. Debouncing, Timers and Interrupts. Asynchronous programming.
10. Analogue to Digital Conversion; Pulse Width Modulation (PWM); Assignment 2 Q&A.

This Week:

11. **LCD Display, sending digital signals to a device.**

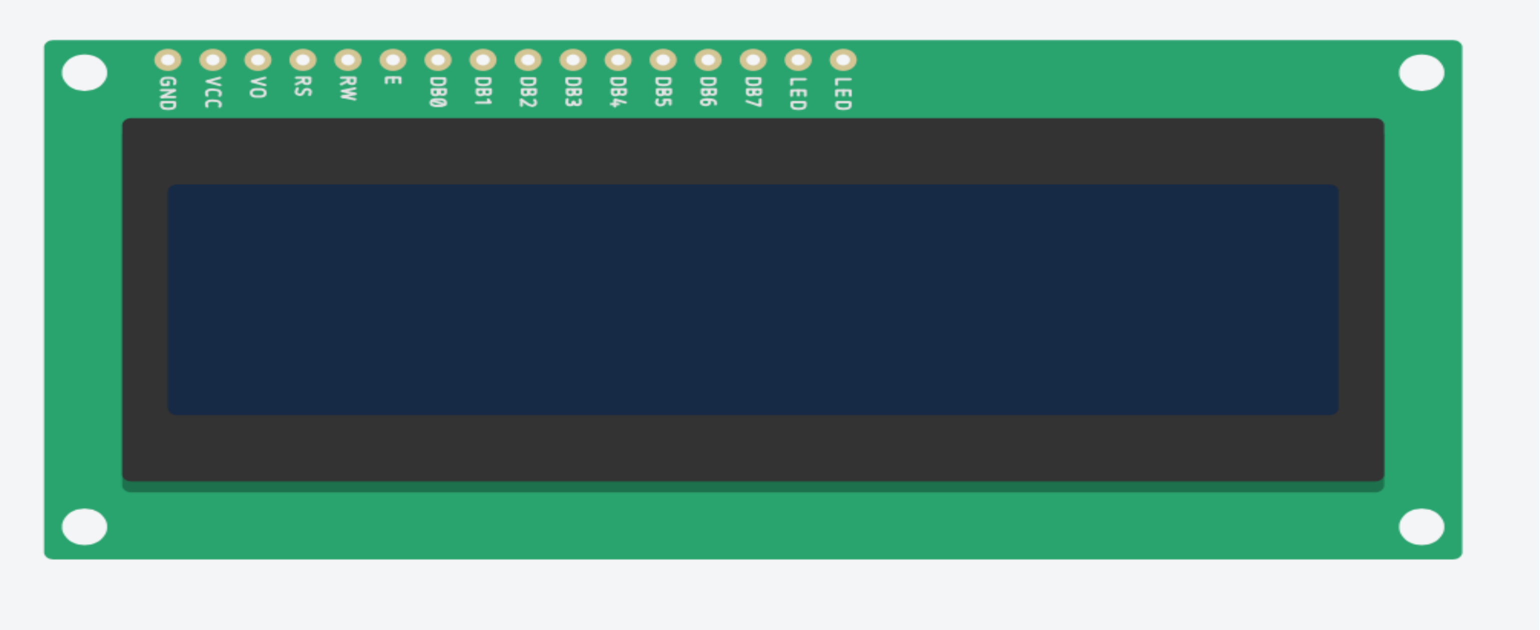
References

Recommended reading:

- Blackboard - Learning Resources - Microcontrollers - LCD datasheet.

The LCD controller/driver

- *Refer: LCD data sheet, p3*.
- Low-power LCD controller.
- 16 characters x 2 lines.
- Build-in back-light with contrast control.
- Interfaces to microcontrollers via using 4 or 8 pins.
  - Data flow is strictly unidirectional, from microcontroller to LCD.
- The controller has a small amount of RAM which holds the data in 8-bit character codes.



LCD Interface (From the LCD Point of View)

LCD has 16 externally accessible pins

- 8 of which are used for data.
- the other pins are the power, control and backlight connections: VCC, GND, RS, R/W, E, etc.
- *Refer: LCD data sheet, section 7.*

Pins we use:

1. **RS -** Register Selector
    - High: DATA. Low: Instruction code.
2. **R/W -** Read and Write mode

  - High: Read (MPU to Module), Low: Write (MPU to Module).

3. **E** - Chip Enable signal
   - From High to Low. Always high.
4. **DB0:7** - Data Input Pin
   - Data is transmitted to the LCD over these pin.
5. **VO** - Contrast control
   - Variable from 0 to VCC.
6. **A/K -** Back-light
   - A = LED + (VCC), K = LED - (GND)
7. **VDD or VCC** - Power
   - Usually 5V

LCD Interface from the ATMega Point of View

The LCD can be connected in a 4-pin or 8-pin mode. Both modes are the same in practices, however the 4-pin mode allows to free-up pins that can be used to connect other devices. We will be using the 4-pin mode.

At the atmega, the LCD pin mappings are:

- Port D, pin 0:7 -> LCD DATA. (in 8-pin mode)
- Port D, pin 4:7 -> LCD DATA. (in 4-pin mode)
- Port B, pin 0 -> LCD E, chip enable pin
- Port B, pin 1 -> LCD RS, register selector

The library provided in these notes allow to select any arbitrary DATA pins in Port D (only), using the following #define

```
// #define LCD_DATA0_PIN (0)
// #define LCD_DATA1_PIN (1)
// #define LCD_DATA2_PIN (2)
// #define LCD_DATA3_PIN (3)
#define LCD_DATA4_PIN (4)
#define LCD_DATA5_PIN (5)
#define LCD_DATA6_PIN (6)
#define LCD_DATA7_PIN (7)
//this configuration uses Port D pins PD4:7 as data pins to the LCD. Pins 0 to 3 aren't used
//as we are working in  4-pin mode


// #define LCD_DATA0_PIN (0)
// #define LCD_DATA1_PIN (1)
// #define LCD_DATA2_PIN (2)
// #define LCD_DATA3_PIN (3)
#define LCD_DATA4_PIN (2)
#define LCD_DATA5_PIN (3)
#define LCD_DATA6_PIN (4)
#define LCD_DATA7_PIN (5)
//this configuration uses Port D pins PD2:5 as data pins to the LCD.
```

The LCD library

The following library can used to send information to the LCD. In tinkedcad there are limitation in the external libraries you can use. The library is an adaptation of the LiquidCrystal library found at https://www.arduino.cc/en/Reference/LiquidCrystal and https://github.com/arduino-libraries/LiquidCrystal

In order for you to use this library on tinkedcad it should be made part of your source code. As follows,

Standard headers used in most examples.

```
#include <avr/io.h>
#include <stdio.h>
#include <string.h>
#include <util/delay.h>
#include <inttypes.h>
```

These definitions are used to configure the DATA pins in Port D. Only 4-pin mode is used here.

```
//For more information about this library please visit:
//http://www.arduino.cc/en/Reference/LiquidCrystal

//and

//https://github.com/arduino-libraries/LiquidCrystal


// --== WIRING ==--
// LCD GND  -> GND
// LCD VCC  -> 5V
// LCD V0   -> GND
// LCD RW   -> GND
// LCD LED Anode    -> 220 Ohm -> 5V
// LCD LED Cathode  -> GND

//Change the values in these defines to reflect
//  how you've hooked up the screen
//In 4-pin mode only DATA4:7 are used

#define LCD_USING_4PIN_MODE (1)

// #define LCD_DATA0_DDR (DDRD)
// #define LCD_DATA1_DDR (DDRD)
// #define LCD_DATA2_DDR (DDRD)
// #define LCD_DATA3_DDR (DDRD)
#define LCD_DATA4_DDR (DDRD)
#define LCD_DATA5_DDR (DDRD)
#define LCD_DATA6_DDR (DDRD)
#define LCD_DATA7_DDR (DDRD)


// #define LCD_DATA0_PORT (PORTD)
// #define LCD_DATA1_PORT (PORTD)
// #define LCD_DATA2_PORT (PORTD)
// #define LCD_DATA3_PORT (PORTD)
#define LCD_DATA4_PORT (PORTD)
#define LCD_DATA5_PORT (PORTD)
#define LCD_DATA6_PORT (PORTD)
#define LCD_DATA7_PORT (PORTD)
```

```
// #define LCD_DATA0_PIN (0)
// #define LCD_DATA1_PIN (1)
// #define LCD_DATA2_PIN (2)
// #define LCD_DATA3_PIN (3)
#define LCD_DATA4_PIN (4)
#define LCD_DATA5_PIN (5)
#define LCD_DATA6_PIN (6)
#define LCD_DATA7_PIN (7)


#define LCD_RS_DDR (DDRB)
#define LCD_ENABLE_DDR (DDRB)

#define LCD_RS_PORT (PORTB)
#define LCD_ENABLE_PORT (PORTB)

#define LCD_RS_PIN (1)
#define LCD_ENABLE_PIN (0)
```

These are specific definitions for the LCD used for control and display

```


// DATASHEET: https://s3-us-west-1.amazonaws.com/123d-circuits-datasheets/
// uploads%2F1431564901240-mni4g6oo875bfbt9-6492779e35179defaf4482c7ac4f9915%2FLCD-WH1602B-TMI.pdf

// commands
#define LCD_CLEARDISPLAY 0x01
#define LCD_RETURNHOME 0x02
#define LCD_ENTRYMODESET 0x04
#define LCD_DISPLAYCONTROL 0x08
#define LCD_CURSORSHIFT 0x10
#define LCD_FUNCTIONSET 0x20
#define LCD_SETCGRAMADDR 0x40
#define LCD_SETDDRAMADDR 0x80

// flags for display entry mode
#define LCD_ENTRYRIGHT 0x00
#define LCD_ENTRYLEFT 0x02
#define LCD_ENTRYSHIFTINCREMENT 0x01
#define LCD_ENTRYSHIFTDECREMENT 0x00

// flags for display on/off control
```

```
#define LCD_DISPLAYON 0x04
#define LCD_DISPLAYOFF 0x00
#define LCD_CURSORON 0x02
#define LCD_CURSOROFF 0x00
#define LCD_BLINKON 0x01
#define LCD_BLINKOFF 0x00

// flags for display/cursor shift
#define LCD_DISPLAYMOVE 0x08
#define LCD_CURSORMOVE 0x00
#define LCD_MOVERIGHT 0x04
#define LCD_MOVELEFT 0x00

// flags for function set
#define LCD_8BITMODE 0x10
#define LCD_4BITMODE 0x00
#define LCD_2LINE 0x08
#define LCD_1LINE 0x00
#define LCD_5x10DOTS 0x04
#define LCD_5x8DOTS 0x00
```

Library function defintions. Some of these functions work on low level sending the the data bits to the LCD, for example write4bits, write8bits, pilseEnable, etc.. Other functions work on a higher level allowing you to send characters and strings to the LCD, for example. write_string, write_char, createChar, etc.

```
void lcd_init(void);
void lcd_write_string(uint8_t x, uint8_t y, char string[]);
void lcd_write_char(uint8_t x, uint8_t y, char val);
void lcd_clear(void);
void lcd_home(void);

void lcd_createChar(uint8_t, uint8_t[]);
void lcd_setCursor(uint8_t, uint8_t);

void lcd_noDisplay(void);
void lcd_display(void);
void lcd_noBlink(void);
void lcd_blink(void);
void lcd_noCursor(void);
void lcd_cursor(void);
void lcd_leftToRight(void);
void lcd_rightToLeft(void);
void lcd_autoscroll(void);
void lcd_noAutoscroll(void);
void scrollDisplayLeft(void);
void scrollDisplayRight(void);

size_t lcd_write(uint8_t);
void lcd_command(uint8_t);

void lcd_send(uint8_t, uint8_t);
void lcd_write4bits(uint8_t);
void lcd_write8bits(uint8_t);
void lcd_pulseEnable(void);

uint8_t _lcd_displayfunction;
uint8_t _lcd_displaycontrol;
uint8_t _lcd_displaymode;
```

Function implementations

```
void lcd_init(void){
  //dotsize
  if (LCD_USING_4PIN_MODE){
    _lcd_displayfunction = LCD_4BITMODE | LCD_1LINE | LCD_5x8DOTS;
  } else {
    _lcd_displayfunction = LCD_8BITMODE | LCD_1LINE | LCD_5x8DOTS;
  }

  _lcd_displayfunction |= LCD_2LINE;

  // RS Pin
  LCD_RS_DDR |= (1 << LCD_RS_PIN);
  // Enable Pin
  LCD_ENABLE_DDR |= (1 << LCD_ENABLE_PIN);

  #if LCD_USING_4PIN_MODE
    //Set DDR for all the data pins
    LCD_DATA4_DDR |= (1 << LCD_DATA4_PIN);
    LCD_DATA5_DDR |= (1 << LCD_DATA5_PIN);
    LCD_DATA6_DDR |= (1 << LCD_DATA6_PIN);
    LCD_DATA7_DDR |= (1 << LCD_DATA7_PIN);

  #else
    //Set DDR for all the data pins
    LCD_DATA0_DDR |= (1 << LCD_DATA0_PIN);
    LCD_DATA1_DDR |= (1 << LCD_DATA1_PIN);
    LCD_DATA2_DDR |= (1 << LCD_DATA2_PIN);
    LCD_DATA3_DDR |= (1 << LCD_DATA3_PIN);
    LCD_DATA4_DDR |= (1 << LCD_DATA4_PIN);
    LCD_DATA5_DDR |= (1 << LCD_DATA5_PIN);
    LCD_DATA6_DDR |= (1 << LCD_DATA6_PIN);
    LCD_DATA7_DDR |= (1 << LCD_DATA7_PIN);
  #endif

  // SEE PAGE 45/46 OF Hitachi HD44780 DATASHEET FOR INITIALIZATION SPECIFICATION!

  // according to datasheet, we need at least 40ms after power rises above 2.7V
  // before sending commands. Arduino can turn on way before 4.5V so we'll wait 50
  _delay_us(50000);
  // Now we pull both RS and Enable low to begin commands (R/W is wired to ground)
  LCD_RS_PORT &= ~(1 << LCD_RS_PIN);
  LCD_ENABLE_PORT &= ~(1 << LCD_ENABLE_PIN);

  //put the LCD into 4 bit or 8 bit mode
  if (LCD_USING_4PIN_MODE) {
    // this is according to the hitachi HD44780 datasheet
    // figure 24, pg 46

    // we start in 8bit mode, try to set 4 bit mode
    lcd_write4bits(0b0111);
    _delay_us(4500); // wait min 4.1ms

    // second try
    lcd_write4bits(0b0111);
    _delay_us(4500); // wait min 4.1ms

    // third go!
    lcd_write4bits(0b0111);
    _delay_us(150);

    // finally, set to 4-bit interface
    lcd_write4bits(0b0010);
  } else {
    // this is according to the hitachi HD44780 datasheet
    // page 45 figure 23

    // Send function set command sequence
    lcd_command(LCD_FUNCTIONSET | _lcd_displayfunction);
    _delay_us(4500);  // wait more than 4.1ms
```

```
    // second try
    lcd_command(LCD_FUNCTIONSET | _lcd_displayfunction);
    _delay_us(150);

    // third go
    lcd_command(LCD_FUNCTIONSET | _lcd_displayfunction);
  }

  // finally, set # lines, font size, etc.
  lcd_command(LCD_FUNCTIONSET | _lcd_displayfunction);

  // turn the display on with no cursor or blinking default
  _lcd_displaycontrol = LCD_DISPLAYON | LCD_CURSOROFF | LCD_BLINKOFF;
  lcd_display();

  // clear it off
  lcd_clear();

  // Initialize to default text direction (for romance languages)
  _lcd_displaymode = LCD_ENTRYLEFT | LCD_ENTRYSHIFTDECREMENT;
  // set the entry mode
  lcd_command(LCD_ENTRYMODESET | _lcd_displaymode);
}


/********** high level commands, for the user! */
void lcd_write_string(uint8_t x, uint8_t y, char string[]){
  lcd_setCursor(x,y);
  for(int i=0; string[i]!='\0'; ++i){
    lcd_write(string[i]);
  }
}

void lcd_write_char(uint8_t x, uint8_t y, char val){
  lcd_setCursor(x,y);
  lcd_write(val);
}

void lcd_clear(void){
  lcd_command(LCD_CLEARDISPLAY);  // clear display, set cursor position to zero
  _delay_us(2000);  // this command takes a long time!
}

void lcd_home(void){
  lcd_command(LCD_RETURNHOME);  // set cursor position to zero
  _delay_us(2000);  // this command takes a long time!
```

```
}


// Allows us to fill the first 8 CGRAM locations
// with custom characters
void lcd_createChar(uint8_t location, uint8_t charmap[]) {
  location &= 0x7; // we only have 8 locations 0-7
  lcd_command(LCD_SETCGRAMADDR | (location << 3));
  for (int i=0; i<8; i++) {
    lcd_write(charmap[i]);
  }
}


void lcd_setCursor(uint8_t col, uint8_t row){
  if ( row >= 2 ) {
    row = 1;
  }

  lcd_command(LCD_SETDDRAMADDR | (col + row*0x40));
}

// Turn the display on/off (quickly)
void lcd_noDisplay(void) {
  _lcd_displaycontrol &= ~LCD_DISPLAYON;
  lcd_command(LCD_DISPLAYCONTROL | _lcd_displaycontrol);
}
void lcd_display(void) {
  _lcd_displaycontrol |= LCD_DISPLAYON;
  lcd_command(LCD_DISPLAYCONTROL | _lcd_displaycontrol);
}

// Turns the underline cursor on/off
void lcd_noCursor(void) {
  _lcd_displaycontrol &= ~LCD_CURSORON;
  lcd_command(LCD_DISPLAYCONTROL | _lcd_displaycontrol);
}
void lcd_cursor(void) {
  _lcd_displaycontrol |= LCD_CURSORON;
  lcd_command(LCD_DISPLAYCONTROL | _lcd_displaycontrol);
}

// Turn on and off the blinking cursor
void lcd_noBlink(void) {
  _lcd_displaycontrol &= ~LCD_BLINKON;
  lcd_command(LCD_DISPLAYCONTROL | _lcd_displaycontrol);
```

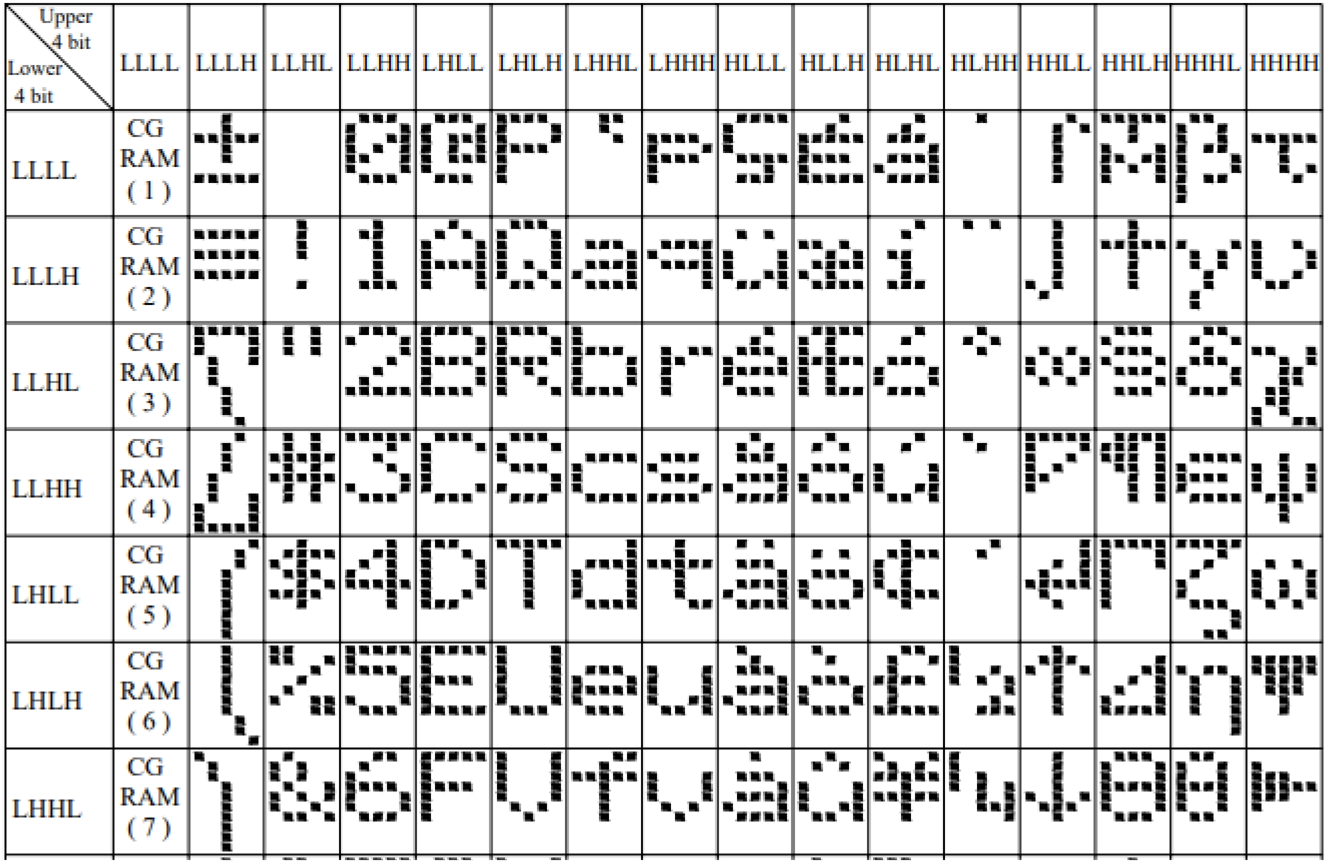
```
}
void lcd_blink(void) {
  _lcd_displaycontrol |= LCD_BLINKON;
  lcd_command(LCD_DISPLAYCONTROL | _lcd_displaycontrol);
}

// These commands scroll the display without changing the RAM
void scrollDisplayLeft(void) {
  lcd_command(LCD_CURSORSHIFT | LCD_DISPLAYMOVE | LCD_MOVELEFT);
}
void scrollDisplayRight(void) {
  lcd_command(LCD_CURSORSHIFT | LCD_DISPLAYMOVE | LCD_MOVERIGHT);
}

// This is for text that flows Left to Right
void lcd_leftToRight(void) {
  _lcd_displaymode |= LCD_ENTRYLEFT;
  lcd_command(LCD_ENTRYMODESET | _lcd_displaymode);
}

// This is for text that flows Right to Left
void lcd_rightToLeft(void) {
  _lcd_displaymode &= ~LCD_ENTRYLEFT;
  lcd_command(LCD_ENTRYMODESET | _lcd_displaymode);
}

// This will 'right justify' text from the cursor
void lcd_autoscroll(void) {
  _lcd_displaymode |= LCD_ENTRYSHIFTINCREMENT;
  lcd_command(LCD_ENTRYMODESET | _lcd_displaymode);
}

// This will 'left justify' text from the cursor
void lcd_noAutoscroll(void) {
  _lcd_displaymode &= ~LCD_ENTRYSHIFTINCREMENT;
  lcd_command(LCD_ENTRYMODESET | _lcd_displaymode);
}

/*********** mid level commands, for sending data/cmds */

inline void lcd_command(uint8_t value) {
  //
  lcd_send(value, 0);
}

inline size_t lcd_write(uint8_t value) {
  lcd_send(value, 1);
  return 1; // assume sucess
}

/************ low level data pushing commands **********/

// write either command or data, with automatic 4/8-bit selection
void lcd_send(uint8_t value, uint8_t mode) {
  //RS Pin
  LCD_RS_PORT &= ~(1 << LCD_RS_PIN);
  LCD_RS_PORT |= (!!mode << LCD_RS_PIN);

  if (LCD_USING_4PIN_MODE) {
    lcd_write4bits(value>>4);
    lcd_write4bits(value);
  } else {
    lcd_write8bits(value);
  }
}

void lcd_pulseEnable(void) {
  //Enable Pin
  LCD_ENABLE_PORT &= ~(1 << LCD_ENABLE_PIN);
  _delay_us(1);
  LCD_ENABLE_PORT |= (1 << LCD_ENABLE_PIN);
  _delay_us(1);    // enable pulse must be >450ns
  LCD_ENABLE_PORT &= ~(1 << LCD_ENABLE_PIN);
  _delay_us(100);   // commands need > 37us to settle
}

void lcd_write4bits(uint8_t value) {
  //Set each wire one at a time

  LCD_DATA4_PORT &= ~(1 << LCD_DATA4_PIN);
  LCD_DATA4_PORT |= ((value & 1) << LCD_DATA4_PIN);
  value >>= 1;

  LCD_DATA5_PORT &= ~(1 << LCD_DATA5_PIN);
  LCD_DATA5_PORT |= ((value & 1) << LCD_DATA5_PIN);
  value >>= 1;

  LCD_DATA6_PORT &= ~(1 << LCD_DATA6_PIN);
  LCD_DATA6_PORT |= ((value & 1) << LCD_DATA6_PIN);
  value >>= 1;

  LCD_DATA7_PORT &= ~(1 << LCD_DATA7_PIN);
  LCD_DATA7_PORT |= ((value & 1) << LCD_DATA7_PIN);

  lcd_pulseEnable();
}

void lcd_write8bits(uint8_t value) {
  //Set each wire one at a time

  #if !LCD_USING_4PIN_MODE
    LCD_DATA0_PORT &= ~(1 << LCD_DATA0_PIN);
    LCD_DATA0_PORT |= ((value & 1) << LCD_DATA0_PIN);
    value >>= 1;

    LCD_DATA1_PORT &= ~(1 << LCD_DATA1_PIN);
    LCD_DATA1_PORT |= ((value & 1) << LCD_DATA1_PIN);
    value >>= 1;

    LCD_DATA2_PORT &= ~(1 << LCD_DATA2_PIN);
    LCD_DATA2_PORT |= ((value & 1) << LCD_DATA2_PIN);
    value >>= 1;

    LCD_DATA3_PORT &= ~(1 << LCD_DATA3_PIN);
    LCD_DATA3_PORT |= ((value & 1) << LCD_DATA3_PIN);
    value >>= 1;

    LCD_DATA4_PORT &= ~(1 << LCD_DATA4_PIN);
    LCD_DATA4_PORT |= ((value & 1) << LCD_DATA4_PIN);
    value >>= 1;

    LCD_DATA5_PORT &= ~(1 << LCD_DATA5_PIN);
    LCD_DATA5_PORT |= ((value & 1) << LCD_DATA5_PIN);
    value >>= 1;

    LCD_DATA6_PORT &= ~(1 << LCD_DATA6_PIN);
    LCD_DATA6_PORT |= ((value & 1) << LCD_DATA6_PIN);
    value >>= 1;

    LCD_DATA7_PORT &= ~(1 << LCD_DATA7_PIN);
    LCD_DATA7_PORT |= ((value & 1) << LCD_DATA7_PIN);

    lcd_pulseEnable();
  #endif
}
```

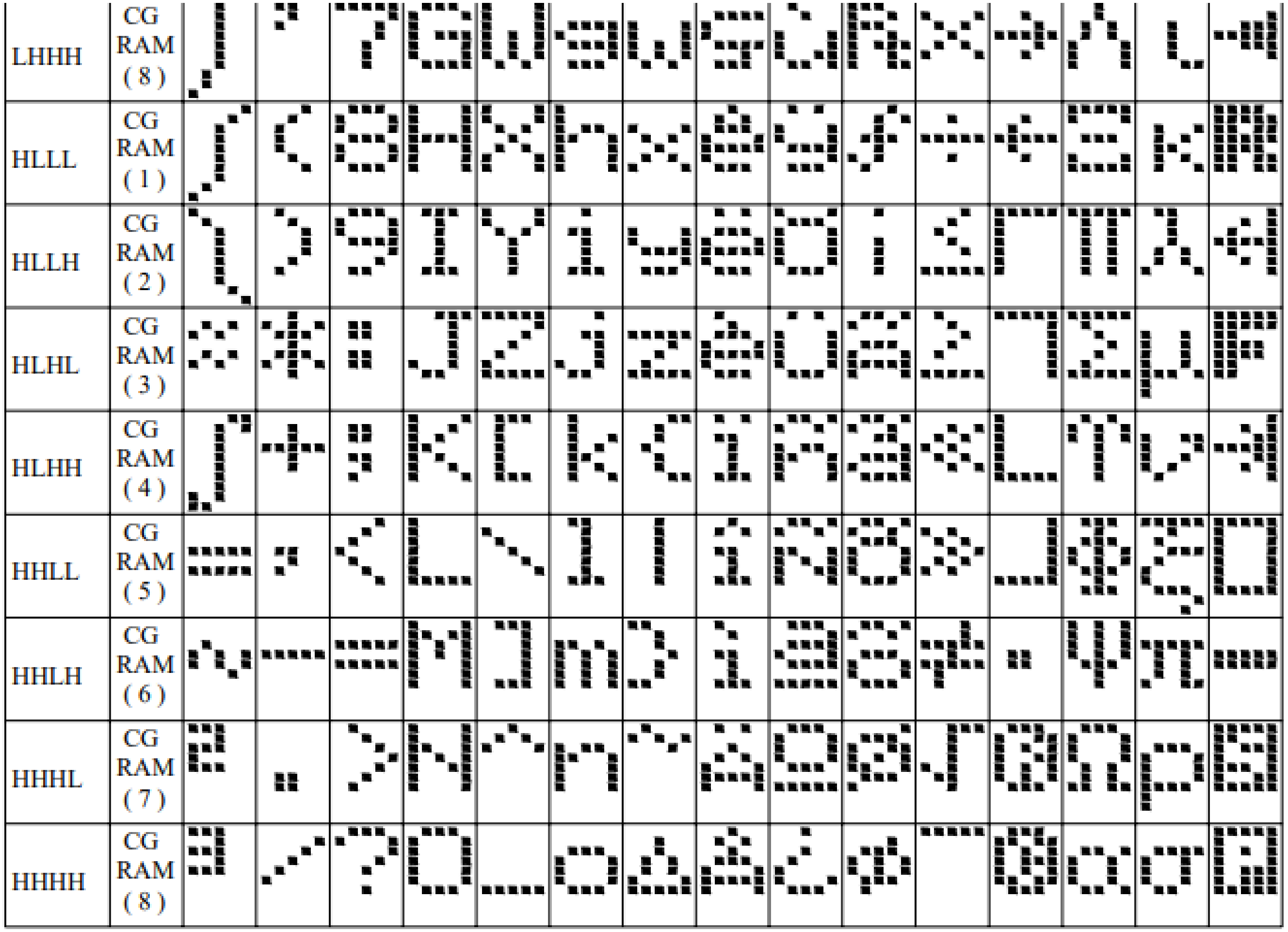
Creating custom characters

We can create custom characters by creating a bitmap first and then using the function **`lcd_createChar`**. You can also write characters from the table below using the function **`lcd_write_char.`**

`For example, the chracter A is LHLLLLLH with L=0, H=1.` upper 4 bits are 7:4 and lower 4 bits are 0:3. Using the function **`lcd_write_char`** one would write the character A on the first row and fourth colunm.

lcd_write_char(4, 0, 0b01000001);


Upper 4 bit
Lower 4 bit
LLLL
LLLH
LLHL
LLHH
LHLL
LHLH
LHHL
LHHH
HLLL
HLLH
HLHL
HLHH
HHLL
HHLH
HHHL
HHHH
LLLL
LLLH
LLHL
LLHH
LHLL
LHLH
LHHL
CG RAM (1)
CG RAM (2)
CG RAM (3)
CG RAM (4)
CG RAM (5)
CG RAM (6)
CG RAM (7)


LHHH
CG RAM (8)
HLLL
CG RAM (1)
HLLH
CG RAM (2)
HLHL
CG RAM (3)
HLHH
CG RAM (4)
HHLL
CG RAM (5)
HHLH
CG RAM (6)
HHHL
CG RAM (7)
HHHH
CG RAM (8)

Custom characters can be create as follows:

Create a bitmap:

```
uint8_t bmp3[8] = { 0b00000000,
                                        0b00011111,
                                        0b00000000,
                                        0b00011111,
                                        0b00000000,
                                        0b00011111,
                                        0b00000000,
                                        0b00011111};

//then call the function
 lcd_createChar(3, bmp3); // assigning this chracted the number 3.

//to write this chracter on the LCD then we call
lcd_write(3);
```

Example: Display characters

```
#include <avr/io.h>
#include <stdio.h>
#include <string.h>
#include <inttypes.h>

//For more information about this library please visit:
//http://www.arduino.cc/en/Reference/LiquidCrystal

//and

//https://github.com/arduino-libraries/LiquidCrystal



// --== WIRING ==--
// LCD GND  -> GND
// LCD VCC  -> 5V
// LCD V0   -> GND
// LCD RW   -> GND
// LCD LED Anode    -> 220 Ohm -> 5V
// LCD LED Cathode  -> GND

//Change the values in these defines to reflect
//  how you've hooked up the screen
//In 4-pin mode only DATA4:7 are used
```

```
#define LCD_USING_4PIN_MODE (1)

// #define LCD_DATA0_DDR (DDRD)
// #define LCD_DATA1_DDR (DDRD)
// #define LCD_DATA2_DDR (DDRD)
// #define LCD_DATA3_DDR (DDRD)
#define LCD_DATA4_DDR (DDRD)
#define LCD_DATA5_DDR (DDRD)
#define LCD_DATA6_DDR (DDRD)
#define LCD_DATA7_DDR (DDRD)


// #define LCD_DATA0_PORT (PORTD)
// #define LCD_DATA1_PORT (PORTD)
// #define LCD_DATA2_PORT (PORTD)
// #define LCD_DATA3_PORT (PORTD)
#define LCD_DATA4_PORT (PORTD)
#define LCD_DATA5_PORT (PORTD)
#define LCD_DATA6_PORT (PORTD)
#define LCD_DATA7_PORT (PORTD)

// #define LCD_DATA0_PIN (0)
// #define LCD_DATA1_PIN (1)
// #define LCD_DATA2_PIN (2)
// #define LCD_DATA3_PIN (3)
#define LCD_DATA4_PIN (4)
#define LCD_DATA5_PIN (5)
#define LCD_DATA6_PIN (6)
#define LCD_DATA7_PIN (7)


#define LCD_RS_DDR (DDRB)
#define LCD_ENABLE_DDR (DDRB)

#define LCD_RS_PORT (PORTB)
#define LCD_ENABLE_PORT (PORTB)

#define LCD_RS_PIN (1)
#define LCD_ENABLE_PIN (0)





//DATASHEET: https://s3-us-west-1.amazonaws.com/123d-circuits-datasheets/uploads%2F1431564901240-mni4g6oo875bfbt9-6492779e35179defaf4482c7ac4f9915%2FLCD-WH1602B-TMI.pdf
```

```
// commands
#define LCD_CLEARDISPLAY 0x01
#define LCD_RETURNHOME 0x02
#define LCD_ENTRYMODESET 0x04
#define LCD_DISPLAYCONTROL 0x08
#define LCD_CURSORSHIFT 0x10
#define LCD_FUNCTIONSET 0x20
#define LCD_SETCGRAMADDR 0x40
#define LCD_SETDDRAMADDR 0x80

// flags for display entry mode
#define LCD_ENTRYRIGHT 0x00
#define LCD_ENTRYLEFT 0x02
#define LCD_ENTRYSHIFTINCREMENT 0x01
#define LCD_ENTRYSHIFTDECREMENT 0x00

// flags for display on/off control
#define LCD_DISPLAYON 0x04
#define LCD_DISPLAYOFF 0x00
#define LCD_CURSORON 0x02
#define LCD_CURSOROFF 0x00
#define LCD_BLINKON 0x01
#define LCD_BLINKOFF 0x00

// flags for display/cursor shift
#define LCD_DISPLAYMOVE 0x08
#define LCD_CURSORMOVE 0x00
#define LCD_MOVERIGHT 0x04
#define LCD_MOVELEFT 0x00

// flags for function set
#define LCD_8BITMODE 0x10
#define LCD_4BITMODE 0x00
#define LCD_2LINE 0x08
#define LCD_1LINE 0x00
#define LCD_5x10DOTS 0x04
#define LCD_5x8DOTS 0x00

void lcd_init(void);
void lcd_write_string(uint8_t x, uint8_t y, char string[]);
void lcd_write_char(uint8_t x, uint8_t y, char val);


void lcd_clear(void);
void lcd_home(void);
void lcd_createChar(uint8_t, uint8_t[]);
void lcd_setCursor(uint8_t, uint8_t);

void lcd_noDisplay(void);
void lcd_display(void);
void lcd_noBlink(void);
void lcd_blink(void);
void lcd_noCursor(void);
void lcd_cursor(void);
void lcd_leftToRight(void);
void lcd_rightToLeft(void);
void lcd_autoscroll(void);
void lcd_noAutoscroll(void);
void scrollDisplayLeft(void);
void scrollDisplayRight(void);

size_t lcd_write(uint8_t);
void lcd_command(uint8_t);


void lcd_send(uint8_t, uint8_t);
void lcd_write4bits(uint8_t);
void lcd_write8bits(uint8_t);
void lcd_pulseEnable(void);

uint8_t _lcd_displayfunction;
uint8_t _lcd_displaycontrol;
uint8_t _lcd_displaymode;

// END Definitions




//=================================================
//=================================================
//=================================================
//                Example code
//=================================================
//=================================================
//=================================================


uint8_t bmp0[8] = { 0b00010010,
                    0b00001010,
                    0b00001001,
                    0b00000100,
                    0b00000100,
                    0b00010010,
                    0b00001010,
                    0b00001001};
uint8_t bmp1[8] = { 0b00010101,
                    0b00010101,
                    0b00010101,
                    0b00010101,
                    0b00010101,
                    0b00010101,
                    0b00010101,
                    0b00010101};
uint8_t bmp2[8] = { 0b00001001,
                    0b00001010,
                    0b00010010,
                    0b00000100,
                    0b00000100,
                    0b00001001,
                    0b00001010,
                    0b00010010};
uint8_t bmp3[8] = { 0b00000000,
                    0b00011111,
                    0b00000000,
                    0b00011111,
                    0b00000000,
                    0b00011111,
                    0b00000000,
                    0b00011111};


void setup_lcd(void);
void loop(void);


int main(void){
  setup_lcd();
  while(1){
    loop();
    _delay_ms(500);
  }
}


void setup_lcd(void) {
  // set up the LCD in 4-pin or 8-pin mode
  lcd_init();


  // Print a message to the LCD
  lcd_write_string(0, 0, "Hello, world!");
  _delay_ms(1000);

  lcd_clear();


  lcd_write_char(4, 0, 0b00010110);
  lcd_write_char(4, 1, 0b00010111);


  lcd_write_string(7,0,"Hello");
  lcd_write_string(7,1, "World");

  //register 4 new character bitmaps as character codes 0-3
  lcd_createChar(0, bmp0);
  lcd_createChar(1, bmp1);
  lcd_createChar(2, bmp2);
  lcd_createChar(3, bmp3);

  lcd_blink();

}


void loop(void) {
  static uint8_t frame = 0;

  //write the custom character bitmaps one at a time to make an animation
  if (frame == 0){
    lcd_setCursor(1,0);
    lcd_write(0);
    lcd_write(1);
    lcd_setCursor(1,1);
    lcd_write(2);
    lcd_write(3);
  } else if (frame == 1) {
    lcd_setCursor(1,0);
    lcd_write(1);
    lcd_write(2);
    lcd_setCursor(1,1);
    lcd_write(3);
    lcd_write(0);
  } else if (frame == 2){
    lcd_setCursor(1,0);
    lcd_write(2);
    lcd_write(3);
    lcd_setCursor(1,1);
    lcd_write(0);
    lcd_write(1);
  } else if (frame == 3){
    lcd_setCursor(1,0);
    lcd_write(3);
    lcd_write(0);
    lcd_setCursor(1,1);
    lcd_write(1);
    lcd_write(2);
  }

  if(frame % 2 == 0){
    scrollDisplayLeft();
  } else {
    scrollDisplayRight();
  }
  frame = (frame + 1) % 4;
}







/* ********************************************/
// START LIBRARY FUNCTIOMNS

void lcd_init(void){
  //dotsize
  if (LCD_USING_4PIN_MODE){
    _lcd_displayfunction = LCD_4BITMODE | LCD_1LINE | LCD_5x8DOTS;
  } else {
    _lcd_displayfunction = LCD_8BITMODE | LCD_1LINE | LCD_5x8DOTS;
  }

  _lcd_displayfunction |= LCD_2LINE;

  // RS Pin
  LCD_RS_DDR |= (1 << LCD_RS_PIN);
  // Enable Pin
  LCD_ENABLE_DDR |= (1 << LCD_ENABLE_PIN);

  #if LCD_USING_4PIN_MODE
    //Set DDR for all the data pins
    LCD_DATA4_DDR |= (1 << LCD_DATA4_PIN);
    LCD_DATA5_DDR |= (1 << LCD_DATA5_PIN);
    LCD_DATA6_DDR |= (1 << LCD_DATA6_PIN);
    LCD_DATA7_DDR |= (1 << LCD_DATA7_PIN);


#else
    //Set DDR for all the data pins
    LCD_DATA0_DDR |= (1 << LCD_DATA0_PIN);
    LCD_DATA1_DDR |= (1 << LCD_DATA1_PIN);
    LCD_DATA2_DDR |= (1 << LCD_DATA2_PIN);
    LCD_DATA3_DDR |= (1 << LCD_DATA3_PIN);
    LCD_DATA4_DDR |= (1 << LCD_DATA4_PIN);
    LCD_DATA5_DDR |= (1 << LCD_DATA5_PIN);
    LCD_DATA6_DDR |= (1 << LCD_DATA6_PIN);
    LCD_DATA7_DDR |= (1 << LCD_DATA7_PIN);
  #endif

  // SEE PAGE 45/46 OF Hitachi HD44780 DATASHEET FOR INITIALIZATION SPECIFICATION!

  // according to datasheet, we need at least 40ms after power rises above 2.7V
  // before sending commands. Arduino can turn on way before 4.5V so we'll wait 50
  _delay_us(50000);
  // Now we pull both RS and Enable low to begin commands (R/W is wired to ground)
  LCD_RS_PORT &= ~(1 << LCD_RS_PIN);
  LCD_ENABLE_PORT &= ~(1 << LCD_ENABLE_PIN);

  //put the LCD into 4 bit or 8 bit mode
  if (LCD_USING_4PIN_MODE) {
    // this is according to the hitachi HD44780 datasheet
    // figure 24, pg 46

    // we start in 8bit mode, try to set 4 bit mode
    lcd_write4bits(0b0111);
    _delay_us(4500); // wait min 4.1ms

    // second try
    lcd_write4bits(0b0111);
    _delay_us(4500); // wait min 4.1ms
    // third go!
    lcd_write4bits(0b0111);
    _delay_us(150);

    // finally, set to 4-bit interface
    lcd_write4bits(0b0010);
  } else {
    // this is according to the hitachi HD44780 datasheet
    // page 45 figure 23

    // Send function set command sequence
    lcd_command(LCD_FUNCTIONSET | _lcd_displayfunction);
    _delay_us(4500);  // wait more than 4.1ms

    // second try
    lcd_command(LCD_FUNCTIONSET | _lcd_displayfunction);
    _delay_us(150);

    // third go
    lcd_command(LCD_FUNCTIONSET | _lcd_displayfunction);
  }

  // finally, set # lines, font size, etc.
  lcd_command(LCD_FUNCTIONSET | _lcd_displayfunction);

  // turn the display on with no cursor or blinking default
  _lcd_displaycontrol = LCD_DISPLAYON | LCD_CURSOROFF | LCD_BLINKOFF;
  lcd_display();

  // clear it off
  lcd_clear();

  // Initialize to default text direction (for romance languages)
  _lcd_displaymode = LCD_ENTRYLEFT | LCD_ENTRYSHIFTDECREMENT;
  // set the entry mode
  lcd_command(LCD_ENTRYMODESET | _lcd_displaymode);
}


/********** high level commands, for the user! */
void lcd_write_string(uint8_t x, uint8_t y, char string[]){
  lcd_setCursor(x,y);
  for(int i=0; string[i]!='\0'; ++i){
    lcd_write(string[i]);
  }
}
```

```c
void lcd_write_char(uint8_t x, uint8_t y, char val){
  lcd_setCursor(x,y);
  lcd_write(val);
}

void lcd_clear(void){
  lcd_command(LCD_CLEARDISPLAY);  // clear display, set cursor position to zero
  _delay_us(2000);  // this command takes a long time!
}

void lcd_home(void){
  lcd_command(LCD_RETURNHOME);  // set cursor position to zero
  _delay_us(2000);  // this command takes a long time!
}


// Allows us to fill the first 8 CGRAM locations
// with custom characters
void lcd_createChar(uint8_t location, uint8_t charmap[]) {
  location &= 0x7; // we only have 8 locations 0-7
  lcd_command(LCD_SETCGRAMADDR | (location << 3));
  for (int i=0; i<8; i++) {
    lcd_write(charmap[i]);
  }
}


void lcd_setCursor(uint8_t col, uint8_t row){
  if ( row >= 2 ) {
    row = 1;
  }

  lcd_command(LCD_SETDDRAMADDR | (col + row*0x40));
}

// Turn the display on/off (quickly)
void lcd_noDisplay(void) {
  _lcd_displaycontrol &= ~LCD_DISPLAYON;
  lcd_command(LCD_DISPLAYCONTROL | _lcd_displaycontrol);
}
void lcd_display(void) {
  _lcd_displaycontrol |= LCD_DISPLAYON;
  lcd_command(LCD_DISPLAYCONTROL | _lcd_displaycontrol);
}
```

```
// Turns the underline cursor on/off
void lcd_noCursor(void) {
  _lcd_displaycontrol &= ~LCD_CURSORON;
  lcd_command(LCD_DISPLAYCONTROL | _lcd_displaycontrol);
}
void lcd_cursor(void) {
  _lcd_displaycontrol |= LCD_CURSORON;
  lcd_command(LCD_DISPLAYCONTROL | _lcd_displaycontrol);
}

// Turn on and off the blinking cursor
void lcd_noBlink(void) {
  _lcd_displaycontrol &= ~LCD_BLINKON;
  lcd_command(LCD_DISPLAYCONTROL | _lcd_displaycontrol);
}
void lcd_blink(void) {
  _lcd_displaycontrol |= LCD_BLINKON;
  lcd_command(LCD_DISPLAYCONTROL | _lcd_displaycontrol);
}

// These commands scroll the display without changing the RAM
void scrollDisplayLeft(void) {
  lcd_command(LCD_CURSORSHIFT | LCD_DISPLAYMOVE | LCD_MOVELEFT);
}
void scrollDisplayRight(void) {
  lcd_command(LCD_CURSORSHIFT | LCD_DISPLAYMOVE | LCD_MOVERIGHT);
}

// This is for text that flows Left to Right
void lcd_leftToRight(void) {
  _lcd_displaymode |= LCD_ENTRYLEFT;
  lcd_command(LCD_ENTRYMODESET | _lcd_displaymode);
}

// This is for text that flows Right to Left
void lcd_rightToLeft(void) {
  _lcd_displaymode &= ~LCD_ENTRYLEFT;
  lcd_command(LCD_ENTRYMODESET | _lcd_displaymode);
}

// This will 'right justify' text from the cursor
void lcd_autoscroll(void) {
  _lcd_displaymode |= LCD_ENTRYSHIFTINCREMENT;
  lcd_command(LCD_ENTRYMODESET | _lcd_displaymode);
}
```

```
// This will 'left justify' text from the cursor
void lcd_noAutoscroll(void) {
  _lcd_displaymode &= ~LCD_ENTRYSHIFTINCREMENT;
  lcd_command(LCD_ENTRYMODESET | _lcd_displaymode);
}

/*********** mid level commands, for sending data/cmds */

inline void lcd_command(uint8_t value) {
  //
  lcd_send(value, 0);
}

inline size_t lcd_write(uint8_t value) {
  lcd_send(value, 1);
  return 1; // assume sucess
}

/************ low level data pushing commands **********/

// write either command or data, with automatic 4/8-bit selection
void lcd_send(uint8_t value, uint8_t mode) {
  //RS Pin
  LCD_RS_PORT &= ~(1 << LCD_RS_PIN);
  LCD_RS_PORT |= (!!mode << LCD_RS_PIN);

  if (LCD_USING_4PIN_MODE) {
    lcd_write4bits(value>>4);
    lcd_write4bits(value);
  } else {
    lcd_write8bits(value);
  }
}

void lcd_pulseEnable(void) {
  //Enable Pin
  LCD_ENABLE_PORT &= ~(1 << LCD_ENABLE_PIN);
  _delay_us(1);
  LCD_ENABLE_PORT |= (1 << LCD_ENABLE_PIN);
  _delay_us(1);    // enable pulse must be >450ns
  LCD_ENABLE_PORT &= ~(1 << LCD_ENABLE_PIN);
  _delay_us(100);   // commands need > 37us to settle
}

void lcd_write4bits(uint8_t value) {
  //Set each wire one at a time
  LCD_DATA4_PORT &= ~(1 << LCD_DATA4_PIN);
  LCD_DATA4_PORT |= ((value & 1) << LCD_DATA4_PIN);
  value >>= 1;

  LCD_DATA5_PORT &= ~(1 << LCD_DATA5_PIN);
  LCD_DATA5_PORT |= ((value & 1) << LCD_DATA5_PIN);
  value >>= 1;

  LCD_DATA6_PORT &= ~(1 << LCD_DATA6_PIN);
  LCD_DATA6_PORT |= ((value & 1) << LCD_DATA6_PIN);
  value >>= 1;

  LCD_DATA7_PORT &= ~(1 << LCD_DATA7_PIN);
  LCD_DATA7_PORT |= ((value & 1) << LCD_DATA7_PIN);

  lcd_pulseEnable();
}

void lcd_write8bits(uint8_t value) {
  //Set each wire one at a time

  #if !LCD_USING_4PIN_MODE
    LCD_DATA0_PORT &= ~(1 << LCD_DATA0_PIN);
    LCD_DATA0_PORT |= ((value & 1) << LCD_DATA0_PIN);
    value >>= 1;

    LCD_DATA1_PORT &= ~(1 << LCD_DATA1_PIN);
    LCD_DATA1_PORT |= ((value & 1) << LCD_DATA1_PIN);
    value >>= 1;

    LCD_DATA2_PORT &= ~(1 << LCD_DATA2_PIN);
    LCD_DATA2_PORT |= ((value & 1) << LCD_DATA2_PIN);
    value >>= 1;

    LCD_DATA3_PORT &= ~(1 << LCD_DATA3_PIN);
    LCD_DATA3_PORT |= ((value & 1) << LCD_DATA3_PIN);
    value >>= 1;

    LCD_DATA4_PORT &= ~(1 << LCD_DATA4_PIN);
    LCD_DATA4_PORT |= ((value & 1) << LCD_DATA4_PIN);
    value >>= 1;

    LCD_DATA5_PORT &= ~(1 << LCD_DATA5_PIN);
    LCD_DATA5_PORT |= ((value & 1) << LCD_DATA5_PIN);
    value >>= 1;
    LCD_DATA6_PORT &= ~(1 << LCD_DATA6_PIN);
    LCD_DATA6_PORT |= ((value & 1) << LCD_DATA6_PIN);
    value >>= 1;

    LCD_DATA7_PORT &= ~(1 << LCD_DATA7_PIN);
    LCD_DATA7_PORT |= ((value & 1) << LCD_DATA7_PIN);

    lcd_pulseEnable();
  #endif
}
```

TinkerCad Version:

https://www.tinkercad.com/things/eX7tJxNIxTF

Summary:

The library provides enough functionality to display characters, strings and custom bitmaps. Additional features include scrolling right and left. We recommend to practice using different functions modifiyng previous examples and displaying information on the LCD. For example, try to display ADC, PWM, timer count and executing time or clock on the LCD.

Additional exercices:

- Try any of the previous examples/exercices and add display functionality. Here is an example to get you started.

https://www.tinkercad.com/things/gwSHlJZZbAT

The End