# LCD in 8bit mode and USART Serial Communication

## Performance Check

_____ (30) LCD operation

_____ (30) Serial Communications

_____ (120) LCD shows the string from Serial Terminal / Cool Term

_____ (20) Submit your code online in Bb

### Objective: The purpose of this lab is for you to

1. use a 2x24 character LCD to display text
2. gain experience in setting up the UART registers on the Atmega2560
3. investigate the asynchronous serial communication
4. display serial terminal inputs in LCD

### Reference:

For reference, use the datasheet for the **Atmega 2560 and LCD** controller that was supplied online in Brightspace.

### Introduction:

For different circuits or systems to exchange information, there are many communication protocols. Data exchange (communications) between controllers can be carried out through two major categories: parallel and serial. In the parallel interface, data are transferred multiple bits at the same time. They usually require Buses of data, across 8 bits, 16 bits, or more. This method can be used for higher speed applications; however, it is at the cost of distance, IO, and hardware. The LCD is a form of parallel communication. Another way of communication is to transmit the information bit by bit (serially). Serial communications are a common form of communication between controllers. Data exchange can be performed using less IO. There are many serial communication protocols, e.g., USB (universal serial bus), Ethernet, SPI, I2C. These communication protocols can be categorized into two major groups: synchronous and asynchronous.

In synchronous mode, the data line(s) has a standard clock signal, whereas asynchronous mode, data is transferred without an external clock signal. Synchronous mode, because of the common clock signal, it is straight forward, and thus it can support faster transfer rate. But this comes at the cost of one extra wire for the clock signal. SPI, I2C are both examples of synchronous serial communications.

In asynchronous mode, there is no external clock signal. It requires less IO pins, but extra care is placed into the reliability and accuracy of the transferred data. GPS modules usually use this type of communication. In asynchronous mode, both controllers are required to have a clock reside at the controller to dictate the communication speed. Before communication can occur, speed, size of the data, error checking scheme had to be agreed upon. The clock inside these controllers should be accurate enough to ensure accurate information is received.

In this lab, you will use both parallel and serial communications. At the end of the lab, your goal is to type a message from a computer (serial) and display the message in the LCD (parallel).

## Before writing your code - PLAN:

1. Set up the hardware. Decide what IO you are going to use. Complete the IO connection table in Table 1 below.

**Table 1: Hardware Connections**

| Function | ATmega2560 Outputs |
|---|---|
| LED0 | |
| LED1 | |
| LED2 | |
| LED3 | |
| LED4 | |
| LED5 | |
| LED6 | |
| LED7 | |
| | |
| **LCD** | |
| RS | |
| RW | |
| EN | |
| D7 – D1 | |

| Function | ATmega2560 Inputs |
|---|---|
| SW0 | |
| SW1 | |
| SW2 | |
| SW3 | |
| SW4 | |
| SW5 | |
| SW6 | |
| | |
| **Communications (USART)** | **ATmega2560 PORT and pin** |
| TX0 | |
| RX0 | |
| | |
| | |
| | |

2. Connect and test the lights and switches and use the debugger to verify the connections before proceeding to next step.
3. Connect LCD according to the diagram below.
4. Below is the wiring diagram. Use a 10kΩ potentiometer for the contrast adjustment.

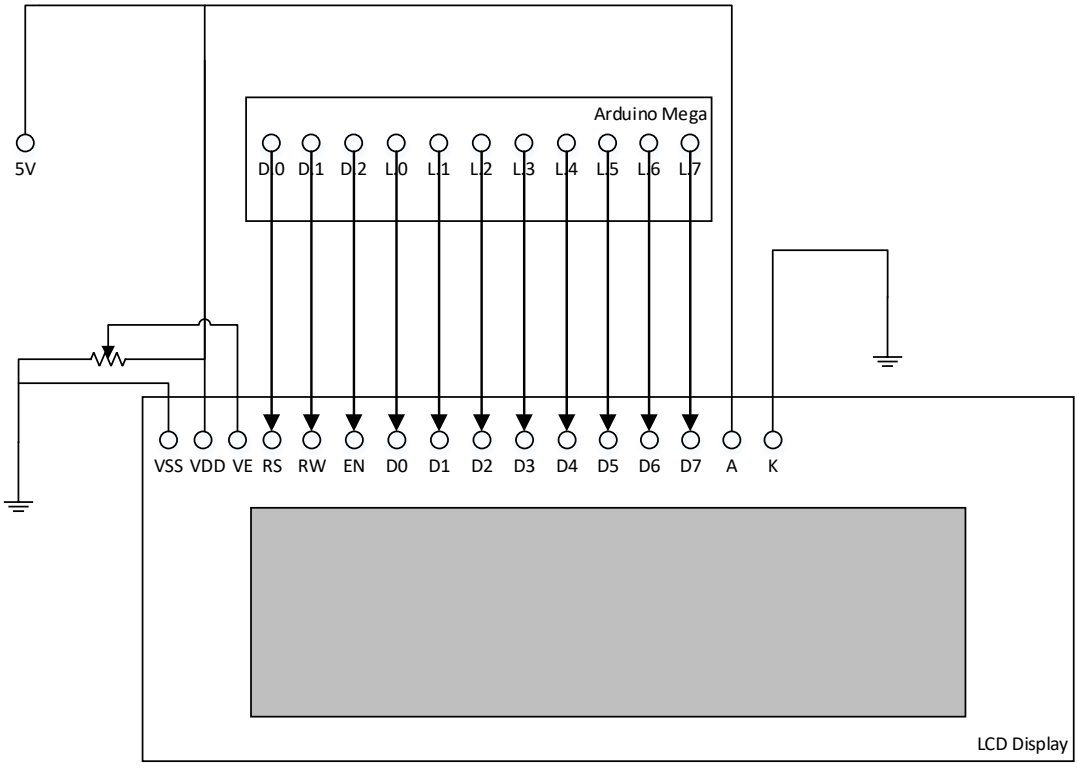


*Figure 1: LCD connections to controller*

1. **Supply voltage should be connected to $V_{DD}$ (+5V) and $V_{SS}$ (GND). The potentiometer controls the contrast.** Make sure that the display contrast potentiometer is turned up sufficiently but NOT too much.
2. Download the test code **LCD_Test_8bit_parallel.c** from Bb and create a project and download the hex file into the ATmega2560. Confirm that the test message is correctly displayed on the LCD to verify that your connections are correct. You should include the test code into your project as a module.

   Test message should read:

   Topline: **Test message:**

   Bottom Line switches every 1 second from: **LCD is OK!**
   to: **8bit parallel**
3. **Convert to LCD module**
   a. Using the supplied test code, convert the project into a multimodule program with a secondary module called LCD.
   b. Add a LCD.c and LCD.h file.
      i. Move all the #define, function prototypes, global variables, and include files that has external access to the .h file
      ii. Move all the functions to the LCD.c file. Leave the main function alone in the main.c
      iii. Include LCD.h in both .c files
      iv. Build the file and fix all the errors.
      v. Program the hex file to your Arduino Mega. It should perform the same as in Step 2.
      vi. Modify your code in the main function, so that you can display this string at the middle of top and bottom rows. “Test”. You may not place space in front of the string e.g. “ Test” for this solution. Consider placing the location at the middle of the top row and the bottom row.
      vii. Set this aside until the last part of the lab.

| **Instructor Check-off** |
|---|
| 1. LCD displays “Test” in the middle of both rows |
| 2. Verify the code to use proper command to display the “Test” message |
| 3. Explain code and demonstrate the use of **modular LCD** code |

**Universal Synchronous Asynchronous serial Receiver and Transmitter (USART)**

4. USART is a flexible serial communication device. The Atmega2560 has four USART’s (USART0 to USART3). Block diagram is included in the Atmel2560 datasheet Figure 22-1. Five registers are used to configure the USART: UCSRA, UCSRB, UCSRC, UBRRH, and UBRRL. The data should be written to the UDR register, then transmitted out of the USART pin. These data (8 data bits at a time) are transmitted out of the TxD pin bit by bit. Information is received from the RxD pin bit by bit. Once 8 data bits have been collected in the UDR register, reading this UDR register can retrieve the data.
5. To ensure accuracy of the transmission and interpretation of the data, the USART has to be configured for its data bits, parity, baud rate.
6. In this exercise, you are to configure the USART as follows:
   - 1 start bit

- 8 data bits
- No parity
- 1 stop bit
- 9600 baud
- USART0 receive pin **R0D** is on PORT _____ ← fill in the blank
- USART0 transmit pin **T0D** is on PORT _____ ← fill in the blank
    - Received data generates an **<u>interrupt</u>**
    - Data transmission uses **<u>polling</u>**

7. Create a module (.c+.h) called USART0. **Leave the LCD module as-is. Remove the code inside the while(1) that call the LCD functions. We will use it later.** Remove all errors.
8. The code in the USART module should have several functions.
    - Initialization of the USART port
    - The Tx routine is in a **function** and uses polling to check when new data can be transmitted.
    - The Rx routine is in an interrupt **ISR** for receipt of new data.
    - Include suitable header comments.
    - Reference lecture notes for sample codes.
9. All the #include, #define, and function prototypes should be located in the .h file.
10. Declaration of global variables that has external linkage should be located in the .h file
11. Functions and ISRs that relate to USART should be located in the USART module
12. All files should have meaningful comments and comment headers. Instructor and Teaching Assistance will not be able to assist you in the lab if you fail to include meaningful comments.
13. In your main code, write the code that displays the variable received from the USART0 and output the switch information to USART0. It should be designed to continuously transmit the bit pattern on switches (SW), receive the bit pattern, and display it on the LEDs. Below are some steps that can aid your design.
14. In the USART module, write a function called init_UART function. This function should have the following:
    a. uint16_t variable called myubr, this variable will store the information that dictate the baud rate
    b. Clear UCSRnA
    c. Review datasheet for UCSRnB in 22.10.3:
        i. set Rx Complete interrupt enable. By setting this bit, whenever you receive a data (1byte) from the RxD pin, the ISR for receive pin will be executed. You do not need to check the UDR for receipt of information periodically in your code.
        ii. Set Receiver Enable bit. This enables the receipt of information from RxD line. Must be set if code is waiting for information serially from another device.
        iii. Set Transmitter Enable. This enables the TxD to send information serially to another controller.
    d. Review datasheet for UCSRnC in 22.10.4:
        i. Set the USART to Asynchronous USART

ii. Disable Parity Mode
iii. 1 stop bit
iv. 8 data bits
v. Clock Polarity to zero

e. myubr equation is located in Table 22-1 using Asynchronous Normal mode. UBRRn is myubr. Write the equation in your code and save it to UBRR3L, then clear UBRR3H.

15. In the USART module, write a function called UART_out (uint8_t ch). This function sends data out of the TxD pin serially, according to the setting you configured in the init_UART function.
    f. This is to use the polling method. Wait for the USART Data Register Empty flag to be set. This flag indicates that the transmit buffer (UDRn) is ready for the next transmission.
    g. Write the variable ch to the transmit buffer (UDRn). Remember to substitute 'n' with your USART number.
16. In the USART module, write an ISR that will be executed whenever the USART Receive Complete interrupt is triggered.
    h. There is only one line in this routine, read the value from the receive buffer (UDRn), save it in a variable called rx_char.
    i. Set up the rx_char variable as a volatile global variable in the USART module. Where should this line be located? _____
17. In the main function, you are to
    j. read from the switches and then send the information to the UART_out function.
    k. Display the value rx_char to LED.
    l. Don't forget to initialize all the ports, #include corresponding .h files, enable global interrupts.
18. Check the operation of your code as follows:

- In loopback mode, the USART3 transmitter output on **TXD0** is connected directly to the receiver **RXD0** on the **same** processor. **Connect a jumper wire between TXD0 and RXD0**. Hence we can check the serial transmission part of the program before connecting the serial port to another device such as another processor. NOTE: remove this jumper wire before loading the code to the controller

  (i) This information in Table 1 should also be included in your comment header. LEDs will be used to demonstrate the ASCII character's binary value, Switches will be used to input the binary value of the ASCII characters
  (ii) Connect the LEDs and the switches according to the hardware connections in Table 1.
  (iii) Connect a jumper wire (loopback jumper) between RXD0 and TXD0 thus forming the loopback on UART0.
  (iv) Press any push buttons, and the corresponding LEDs should light.
  (v) When remove the loopback jumper, pushing a pushbutton and the corresponding LED should not turn on.

**Instructor Check-off**

1. Demonstrate the use of USART module
2. All 8 pushbuttons, and 8 LEDs should be connected for this check off. LED shows the corresponding switch / pushbutton status when loopback jumper is connected. When loopback jumper is removed, LEDs status and pushbuttons status do not correspond to each other

**Serial Stream Capture, Store in ATmega2560 using Rx Interrupts and then to LCD display:**

19. Connect the controller to serial display. **Disconnect the serial loop back jumper**. Control the input push button as 0x41 and see a character A in the serial monitor.

20. Combining it all together: you have some basic structures completed. This is the time to add more functions to make all the modules work together.
    a. Review the flow chart
    b. Add a main.h file for your primary module, then change the #include file in the main.c file from LCD.h to main.h. Be-sure to include LCD.h inside the main.h file.
    c. In the main.h file, add volatile global variables for the **array index, LCD_update,** and also for an array used to store the text received through the USART called **rx_buffer[]**. The rx_buffer is to be displayed on the LCD. Set this array length to a suitable value e.g. 25. You must set the array size.
    d. You should have the LCD module and the USART module in this project.
    e. Modify the code based on the flowcharts at the end of this lab to include the USART Rx ISR to receive and store the ASCII string from CoolTerm or Serial Monitor, and the code in the while(1) loop to display the ASCII string on the LCD.
    f. The ASCII string should start display on the middle of the top row and bottom row

**Instructor Check Off**

1. Demonstrate the use of both LCD and USART modules
2. Explain the flow of data from keyboard to the LCD using the code
3. Type a string in the keyboard, the string display in LCD without extra characters in the display.

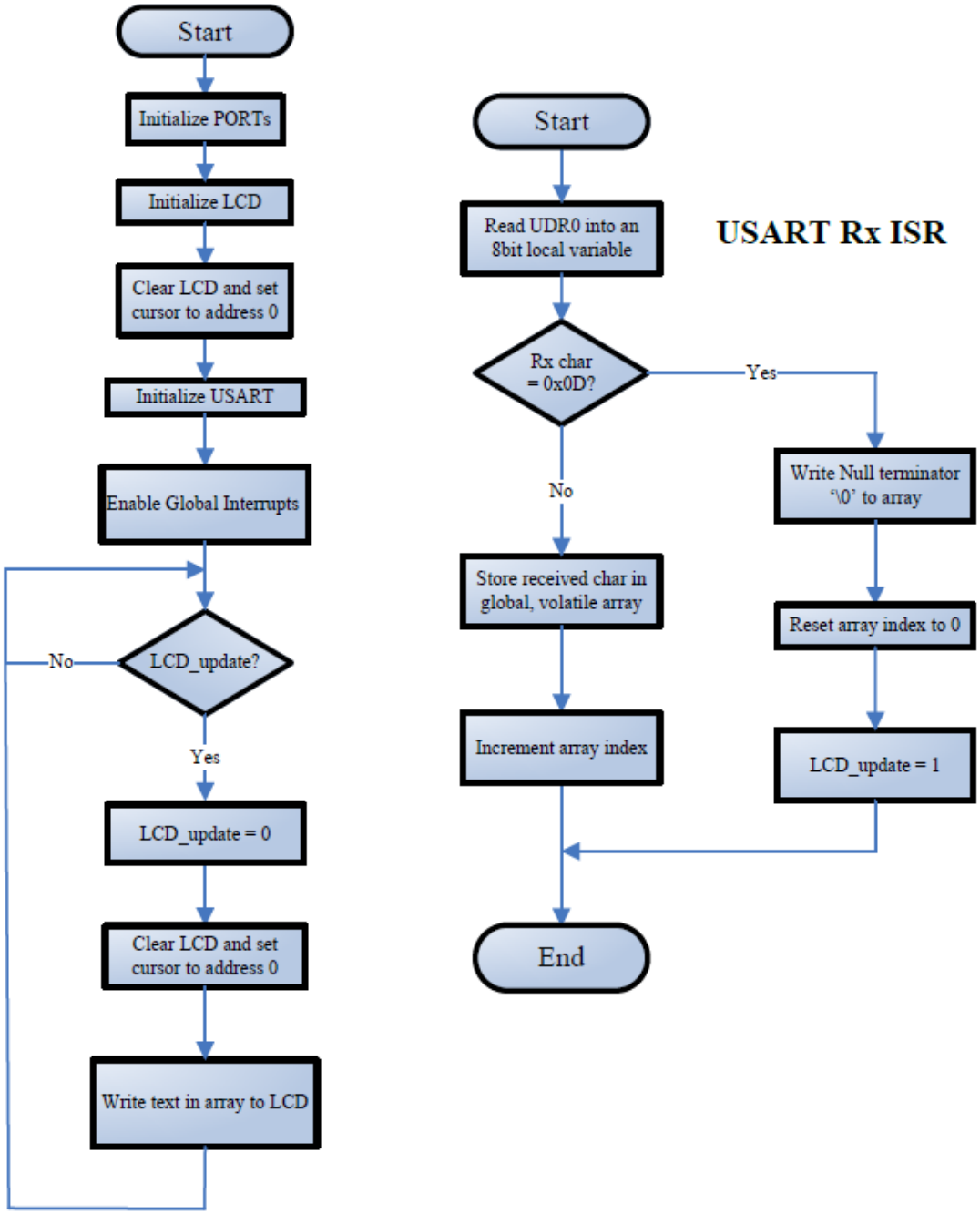


*Figure 2: Flow chart for the last procedure*