**A Detail Explanation of TASK\_16 (RTC ds\_1302 interfacing with 8051)**

Let's break down the code in **LCD.c** line by line, explaining each detail, including the purpose of hexadecimal values, control signals, and how the LCD operates.

**1) LCD.c Detailed Explanation**

**Header Files and Definitions**

#include <reg51.h>

#include "LCD.h"

#include <string.h>

* **#include <reg51.h>**: This header file is specific to the 8051 microcontroller family. It includes the necessary definitions and functions to interact with the 8051's hardware, such as ports and special function registers.
* **#include "LCD.h"**: This includes the LCD.h file, which contains function prototypes and definitions used for LCD operations. It's a custom header file you've created for LCD interfacing.
* **#include <string.h>**: This standard C library is included for string manipulation functions, though it isn't actively used in the provided code.

**Define Data Bus and Control Pins**

#define lcd\_data P0

sbit RW = P2^5;

sbit RS = P2^6;

sbit EN = P2^7;

* **#define lcd\_data P0**: This defines lcd\_data as port P0. The LCD’s data bus (D0-D7) is connected to the 8051’s port P0. When you send commands or data to the LCD, you’ll use this port.

|  |
| --- |
| C:\Users\lenovo\AppData\Local\Packages\Microsoft.Windows.Photos_8wekyb3d8bbwe\TempState\ShareServiceTempFolder\Untitled.jpeg |

* **sbit RW = P2^5;**: This defines the RW pin (Read/Write) as bit 5 of port P2. The RW pin on the LCD controls whether you’re reading from (RW=1) or writing to (RW=0) the LCD.
* **sbit RS = P2^6;**: This defines the RS pin (Register Select) as bit 6 of port P2. The RS pin selects between the command register (RS=0) and data register (RS=1). Commands control the LCD’s operation, while data represents the characters you want to display.
* **sbit EN = P2^7;**: This defines the EN pin (Enable) as bit 7 of port P2. The EN pin is used to latch the data or command into the LCD. A high-to-low transition on this pin tells the LCD to execute the instruction on its data bus.

#### ****LCD Initialization****

void lcd\_init() {

lcd\_command(0x38); // Function Set: 8-bit, 2-line, 5x7 dots

lcd\_command(0x0C); // Display ON, Cursor OFF, Blink OFF

lcd\_command(0x01); // Clear Display

lcd\_command(0x80); // Set cursor to first line, first position

}

* **lcd\_command(0x38)**: This sends the command 0x38 to the LCD. Let's break down what this hexadecimal value means:
  + **0x38** in binary is 0011 1000.
  + The LCD’s Function Set command has the format 001 DL N F x x.
    - **DL=1**: This sets the interface to 8-bit mode, meaning the data is sent on all 8 data pins (D0-D7).
    - **N=1**: This enables 2-line display mode, which allows for two lines of text.
    - **F=0**: This sets the character font to 5x7 dots.
  + So, 0x38 configures the LCD for 8-bit communication, 2-line display, and 5x7 dot characters.
* **lcd\_command(0x0C)**: This sends the command 0x0C to the LCD.
  + **0x0C** in binary is 0000 1100.
  + The Display Control command has the format 0000 1DCB.
    - **D=1**: Display is ON.
    - **C=0**: Cursor is OFF.
    - **B=0**: Blink is OFF.
  + So, 0x0C turns the display on, hides the cursor, and disables blinking.
* **lcd\_command(0x01)**: This sends the command 0x01 to the LCD.
  + **0x01** in binary is 0000 0001.
  + The Clear Display command clears the entire display and sets the cursor position to the top-left corner (address 0x00).
* **lcd\_command(0x80)**: This sends the command 0x80 to the LCD.
  + **0x80** in binary is 1000 0000.
  + The Set DDRAM Address command sets the cursor to the first position of the first line (address 0x00). DDRAM stands for Display Data RAM, where the characters are stored in the LCD.

#### ****Sending a Command to the LCD****

void lcd\_command(unsigned char cmd) {

lcd\_data = cmd; // Put the command on the data bus (P0)

RW = 0; // Set RW to 0, indicating a write operation

RS = 0; // Set RS to 0, selecting the instruction register

EN = 1; // Enable the LCD to read the data/command

delay(1); // Small delay to allow the command to be processed

EN = 0; // Disable the enable pin, latching the command

delay(2); // Additional delay to ensure the command is fully executed

}

* **lcd\_data = cmd;**: The command is placed on the data bus (P0). This bus is connected to the LCD's data pins (D0-D7).
* **RW = 0;**: The RW pin is set to 0 to indicate a write operation. If it were set to 1, it would indicate a read operation.
* **RS = 0;**: The RS pin is set to 0 to select the command register. Commands control the LCD's functions, as opposed to data (characters) that would be displayed on the screen.
* **EN = 1;**: The EN pin is set to 1. This tells the LCD to read the data present on the data bus.
* **delay(1);**: A small delay is introduced to give the LCD enough time to read and process the command.
* **EN = 0;**: The EN pin is set back to 0. This latches the command into the LCD, meaning the LCD will now execute the command.
* **delay(2);**: Another delay ensures the command is fully executed before any further commands or data are sent.

#### ****Displaying a Character on the LCD****

void lcd\_char(unsigned char dat) {

lcd\_data = dat; // Put the character on the data bus (P0)

RW = 0; // Set RW to 0, indicating a write operation

RS = 1; // Set RS to 1, selecting the data register

EN = 1; // Enable the LCD to read the data (character)

delay(100); // Small delay to allow the character to be displayed

EN = 0; // Disable the enable pin, latching the data

delay(200); // Additional delay to ensure the data is fully displayed

}

* **lcd\_data = dat;**: The character data to be displayed is placed on the data bus (P0).
* **RW = 0;**: The RW pin is set to 0 for a write operation.
* **RS = 1;**: The RS pin is set to 1 to select the data register. This tells the LCD that the upcoming data should be displayed as a character.
* **EN = 1;**: The EN pin is set to 1, enabling the LCD to read the data on the bus.
* **delay(100);**: A delay is introduced to ensure the character is properly displayed. The longer delay here compared to commands is because rendering a character can take slightly longer.
* **EN = 0;**: The EN pin is set back to 0, latching the data into the LCD.
* **delay(200);**: An additional delay ensures that the character is fully processed and displayed on the screen before the next operation.

#### ****Displaying a String on the LCD****

void lcd\_string(unsigned char \*str) {

int i;

for(i = 0; str[i] != 0; i++) { // Loop through each character in the string

lcd\_char(str[i]); // Display each character using the lcd\_char function

}

}

* **void lcd\_string(unsigned char \*str)**: This function takes a string (array of characters) as input.
* **for(i = 0; str[i] != 0; i++)**: The loop iterates through each character of the string until it reaches the null terminator (0), which signifies the end of the string.
* **lcd\_char(str[i]);**: For each character in the string, the lcd\_char function is called to display it on the LCD. This function handles the actual communication with the LCD to render each character.

#### ****Setting the Cursor Position****

void lcd\_set\_cursor(unsigned char row, unsigned char col) {

unsigned char position;

if(row == 0) {

position = 0x80 + col; // Set cursor to the specified column on the first line

} else {

position = 0xC0 + col; // Set cursor to the specified column on the second line

}

lcd\_command(position); // Send the calculated position command to the LCD

}

* **void lcd\_set\_cursor(unsigned char row, unsigned char col)**: This function sets the cursor to a specific position on the LCD, defined by the row and col parameters.
* **unsigned char position;**: A variable to store the calculated position.
* **if(row == 0)**: If the desired row is the first line (row 0):
  + **position = 0x80 + col;**: The position is calculated by adding the column number to 0x80. 0x80 is the base address for the first line of the LCD. Adding the column offset sets the exact position within that line.
* **else**: If the desired row is the second line (row 1):
  + **position = 0xC0 + col;**: The position is calculated by adding the column number to 0xC0. 0xC0 is the base address for the second line of the LCD.
* **lcd\_command(position);**: The calculated position command is sent to the LCD, setting the cursor to the specified row and column.

#### ****Delay Function****

void delay(unsigned int time) {

unsigned int i, j;

for(i = 0; i < time; i++) // Outer loop for the delay

for(j = 0; j < 120; j++); // Inner loop for the delay (adjustable for timing)

}

* **void delay(unsigned int time)**: This function creates a delay to give the LCD time to process commands or display data.
* **for(i = 0; i < time; i++)**: The outer loop runs for the specified time value.
* **for(j = 0; j < 120; j++);**: The inner loop runs 120 times for each iteration of the outer loop. This nested loop structure provides a relatively adjustable delay. The actual time delay will depend on the clock speed of the microcontroller and can be tuned as needed.

### ****LCD.h Detailed Explanation****

This file contains the function prototypes and prevents multiple inclusions of the same header file.

#ifndef LCD\_H // Include guard to prevent multiple inclusions

#define LCD\_H

// Function prototypes

void lcd\_init(); // Initialize the LCD

void lcd\_command(unsigned char cmd); // Send a command to the LCD

void lcd\_char(unsigned char dat); // Display a single character on the LCD

void lcd\_string(unsigned char \*str); // Display a string on the LCD

void delay(unsigned int time); // Create a delay

void lcd\_set\_cursor(unsigned char row, unsigned char col); // Set cursor position on the LCD

#endif

* **#ifndef LCD\_H** and **#define LCD\_H**: These lines create an include guard, which prevents the header file from being included multiple times in the same file. This avoids potential redefinition errors.
* **Function Prototypes**: Each function used in LCD.c is declared here, making them available to other files that include LCD.h. The prototypes tell the compiler about the functions' names, return types, and parameters without providing the full implementation.

### ****Summary of Key Concepts****

* **Hexadecimal Commands**: The commands sent to the LCD (like 0x38, 0x0C, etc.) control the LCD's configuration, display settings, and cursor position. These values are standardized for LCD modules that follow the Hitachi HD44780U controller specification.
* **Control Signals**: The RW, RS, and EN pins are critical for the LCD to distinguish between commands and data, as well as when to latch and execute those instructions.
* **Timing Delays**: Delays are necessary because the LCD takes time to execute commands and update the display. Without proper delays, the LCD might miss commands or display incorrect data.

By understanding each part of this code and how it interacts with the LCD, you can control the display effectively and adapt the code to different needs.