main.c

|  |
| --- |
| /\* https://blog.csdn.net/m0\_74068921/article/details/131745259 \*/ |
|  |
|  |
| #include <reg52.h> |
| #include "Ds1302.h" |
| #include "LCD1602.h" |
|  |
| void main() { |
| LCD\_Init(); |
| DS1302\_Init(); |
|  |
| // Initialize the LCD with labels and placeholders |
| LCD\_ShowString(1, 1, "Date: "); |
| LCD\_ShowString(2, 1, "Time: "); |
|  |
| DS1302\_SetTime(); |
|  |
| while (1) { |
| DS1302\_ReadTime(); |
|  |
| // Display Date |
| LCD\_ShowNum(1, 7, DS1302\_Time[2], 2); // Day |
| LCD\_ShowChar(1, 9, '/'); |
| LCD\_ShowNum(1, 10, DS1302\_Time[1], 2); // Month |
| LCD\_ShowChar(1, 12, '/'); |
| LCD\_ShowNum(1, 13, DS1302\_Time[0], 2); // Year |
|  |
| // Display Time |
| LCD\_ShowNum(2, 7, DS1302\_Time[3], 2); // Hour |
| LCD\_ShowChar(2, 9, ':'); |
| LCD\_ShowNum(2, 10, DS1302\_Time[4], 2); // Minute |
| LCD\_ShowChar(2, 12, ':'); |
| LCD\_ShowNum(2, 13, DS1302\_Time[5], 2); // Second |
| } |
| } |
|  |
|  |
|  |
| //#include <reg52.H> |
| //#include "Ds1302.h" |
| //#include "LCD1602.h" |
|  |
| //void main() { |
| // LCD\_Init(); |
| // DS1302\_Init(); |
| // LCD\_ShowString(1, 1, " / / "); |
| // LCD\_ShowString(2, 1, " : : "); |
| // DS1302\_SetTime(); |
|  |
| // while (1) { |
| // DS1302\_ReadTime(); |
|  |
| // LCD\_ShowNum(1, 1, DS1302\_Time[0], 2); |
| // LCD\_ShowNum(1, 4, DS1302\_Time[1], 2); |
| // LCD\_ShowNum(1, 7, DS1302\_Time[2], 2); |
| // LCD\_ShowNum(2, 1, DS1302\_Time[3], 2); |
| // LCD\_ShowNum(2, 4, DS1302\_Time[4], 2); |
| // LCD\_ShowNum(2, 7, DS1302\_Time[5], 2); |
| // } |
| //} |

Let's break down the two functions, DS1302\_Init() and LCD\_Init(), step by step.

**1. DS1302\_Init()**

This function initializes the DS1302 Real-Time Clock (RTC) module. Here’s a detailed explanation of each line:

**void DS1302\_Init()**

**{**

**DS1302\_CE = 0;**

**DS1302\_SCLK = 0;**

**DS1302\_IO = 0; // Set IO to low as default**

**}**

* **DS1302\_CE = 0;**
  + **Purpose:** This line sets the Chip Enable (CE) pin of the DS1302 to low (0 volts). The CE pin is crucial because it controls whether the DS1302 is active or not.
  + **Explanation:** When the CE pin is low, the DS1302 is in a standby mode, meaning it’s not active and won't respond to any commands. By setting this pin to low initially, we ensure that the DS1302 remains inactive until we explicitly enable it later when sending or receiving data.
* **DS1302\_SCLK = 0;**
  + **Purpose:** This line sets the Serial Clock (SCLK) pin of the DS1302 to low.
  + **Explanation:** The SCLK pin is used to synchronize data transfer between the microcontroller and the DS1302. Setting it to low initially ensures that no unintended clock pulses are sent to the DS1302. We will toggle this pin high and low to "clock" the data in or out of the DS1302 during communication.
* **DS1302\_IO = 0;**
  + **Purpose:** This line sets the I/O pin of the DS1302 to low.
  + **Explanation:** The I/O pin is used for data transfer between the DS1302 and the microcontroller. Setting it to low as a default ensures that the pin is in a known state before any data transfer begins. This is important to avoid any unwanted noise or glitches on the line, which could corrupt the data.

**2. LCD\_Init()**

This function initializes a 16x2 LCD display, preparing it to display characters. Let’s go through it step by step:

**void LCD\_Init() {**

**LCD\_WriteCommand(0x38);**

**LCD\_WriteCommand(0x0c);**

**LCD\_WriteCommand(0x06);**

**LCD\_WriteCommand(0x01);**

**}**

* **LCD\_WriteCommand(0x38);**
  + **Purpose:** This command sets the LCD to 8-bit mode, 2 lines, and 5x7 dots per character.
  + **Explanation:** The 0x38 is a hexadecimal command that translates to a specific set of instructions for the LCD. The 8-bit mode tells the LCD to expect 8-bit data for commands and characters. The "2 lines" setting configures the LCD to use both rows of the display. The "5x7 dots" means each character is represented in a 5x7 pixel grid.
* **LCD\_WriteCommand(0x0c);**
  + **Purpose:** This command turns on the display and disables the cursor and blinking.
  + **Explanation:** The 0x0c command tells the LCD to activate the display (so characters will be visible) but not to show a blinking cursor or an underline cursor. It essentially sets the display mode to show text in a clean, non-distracting way.
* **LCD\_WriteCommand(0x06);**
  + **Purpose:** This command sets the LCD to automatically move the cursor to the right after each character is displayed.
  + **Explanation:** The 0x06 command sets the entry mode for the LCD. It tells the display that after writing a character, the cursor should automatically move one position to the right. This is the standard behavior for writing text from left to right.
* **LCD\_WriteCommand(0x01);**
  + **Purpose:** This command clears the display and returns the cursor to the home position.
  + **Explanation:** The 0x01 command is essentially a "clear screen" command. It erases any characters currently on the display and moves the cursor back to the starting position (top left corner). It also resets any custom settings to their defaults. This ensures that the LCD starts with a clean slate, ready to display new data.

**Summary:**

* **DS1302\_Init()** prepares the DS1302 RTC by setting its control pins to a known, inactive state before communication begins.
* **LCD\_Init()** prepares the LCD by configuring it to use 8-bit data, setting up display parameters, and clearing the screen, ensuring it’s ready to show characters correctly.

The lines of code you're asking about are responsible for initializing the LCD with the labels "Date: " and "Time: " on the display. These labels are static and are placed at specific locations on the LCD screen. Let's break down how these lines work in the context of the LCD\_ShowString function.

### Code Lines in Main Function:

LCD\_ShowString(1, 1, "Date: ");

LCD\_ShowString(2, 1, "Time: ");

### Context and Detailed Explanation:

#### 1. **Function Call Overview:**

* **LCD\_ShowString(1, 1, "Date: ");**
  + This line calls the LCD\_ShowString function to display the string "Date: " on the first line (Line 1) of the LCD, starting at the first column (Column 1).
* **LCD\_ShowString(2, 1, "Time: ");**
  + This line calls the LCD\_ShowString function to display the string "Time: " on the second line (Line 2) of the LCD, starting at the first column (Column 1).

#### 2. **Breaking Down the** LCD\_ShowString **Function:**

The function LCD\_ShowString takes three parameters:

* **unsigned char Line:** The line number on the LCD where the string should be displayed. In the first call, this is 1, and in the second, it is 2.
* **unsigned char Column:** The starting column on the LCD where the string should be displayed. For both calls, this is 1.
* **char \*String:** The string to be displayed, which is "Date: " in the first call and "Time: " in the second.

#### 3. **Inside the** LCD\_ShowString **Function:**

void LCD\_ShowString(unsigned char Line, unsigned char Column, char \*String)

{

unsigned char i;

LCD\_SetCursor(Line, Column);

for (i = 0; String[i] != '\0'; i++)

{

LCD\_WriteData(String[i]);

}

}

* **LCD\_SetCursor(Line, Column);**
  + The function first calls LCD\_SetCursor, which positions the cursor on the LCD screen to the specified line and column. This is crucial because it sets the exact location where the string will begin to be displayed.
  + For example, when you call LCD\_ShowString(1, 1, "Date: ");, the cursor is set to the first line and the first column.
* **for (i = 0; String[i] != '\0'; i++)**
  + This loop iterates through each character of the string until it reaches the null terminator (\0), which marks the end of the string.
  + In the loop, String[i] gives the current character of the string at position i.
* **LCD\_WriteData(String[i]);**
  + Inside the loop, LCD\_WriteData is called for each character of the string. This function sends the character to the LCD to be displayed at the current cursor position.
  + The cursor automatically moves to the next position after each character is written, so the next character is displayed in the next column.
* **How It Works for "Date: ":**
  + When "Date: " is passed, the loop will execute for each character (D, a, t, e, :, and a space ), and each character will be written to the LCD starting from the first column of the first line.
  + After this, "Date: " appears on the first line of the LCD.
* **How It Works for "Time: ":**
  + Similarly, when "Time: " is passed, the loop writes each character (T, i, m, e, :, and a space ) starting from the first column of the second line, making "Time: " appear on the second line of the LCD.

### Summary:

The two lines in the main function initialize the LCD by placing the labels "Date: " and "Time: " in their respective positions. This is done by the LCD\_ShowString function, which sets the cursor position on the LCD and then writes each character of the string to the display, ensuring that the labels are correctly positioned on the LCD screen for further updates.

The line DS1302\_SetTime(); is a function call that executes the DS1302\_SetTime() function. This function is responsible for setting the time on the DS1302 Real-Time Clock (RTC) module. Let's go through each line of the DS1302\_SetTime() function in detail to understand how it works.

### Overview of DS1302\_SetTime()

The DS1302\_SetTime() function writes the time and date values to the DS1302 RTC registers. The function uses a helper function DS1302\_WriteByte() to write specific values to the RTC's internal registers.

### Detailed Explanation of Each Line

c

Copy code

void DS1302\_SetTime()

{

DS1302\_WriteByte(DS1302\_WP, 0x00);

1. **Purpose**: The first line of the function disables the write protection on the DS1302.
2. **How it works**: The DS1302\_WriteByte(DS1302\_WP, 0x00); call writes 0x00 to the write protection (WP) register (DS1302\_WP). By setting the WP register to 0x00, the DS1302 is allowed to modify the time and date values. If this step is not done, the DS1302 will prevent any modifications to its time registers.

c

Copy code

DS1302\_WriteByte(DS1302\_YEAR, DS1302\_Time[0] /10\*16 + DS1302\_Time[0]%10);

1. **Purpose**: This line sets the year on the DS1302.
2. **How it works**:
   * DS1302\_Time[0] contains the year value (e.g., 23 for 2023).
   * The expression DS1302\_Time[0] /10\*16 + DS1302\_Time[0]%10 converts the year from decimal format to Binary-Coded Decimal (BCD) format. BCD is a way of storing two digits in a single byte where each digit is stored in a 4-bit nibble.
   * The DS1302\_WriteByte(DS1302\_YEAR, ...) function writes this BCD value to the year register of the DS1302.

c

Copy code

DS1302\_WriteByte(DS1302\_MONTH, DS1302\_Time[1] /10\*16 + DS1302\_Time[1]%10);

1. **Purpose**: This line sets the month on the DS1302.
2. **How it works**:
   * DS1302\_Time[1] contains the month value (e.g., 08 for August).
   * The expression DS1302\_Time[1] /10\*16 + DS1302\_Time[1]%10 converts the month from decimal to BCD format.
   * The DS1302\_WriteByte(DS1302\_MONTH, ...) function writes this BCD value to the month register of the DS1302.

c

Copy code

DS1302\_WriteByte(DS1302\_DATE, DS1302\_Time[2] /10\*16 + DS1302\_Time[2]%10);

1. **Purpose**: This line sets the date on the DS1302.
2. **How it works**:
   * DS1302\_Time[2] contains the date value (e.g., 26 for the 26th day of the month).
   * The expression DS1302\_Time[2] /10\*16 + DS1302\_Time[2]%10 converts the date from decimal to BCD format.
   * The DS1302\_WriteByte(DS1302\_DATE, ...) function writes this BCD value to the date register of the DS1302.

c

Copy code

DS1302\_WriteByte(DS1302\_HOUR, DS1302\_Time[3] /10\*16 + DS1302\_Time[3]%10);

1. **Purpose**: This line sets the hour on the DS1302.
2. **How it works**:
   * DS1302\_Time[3] contains the hour value (e.g., 15 for 3 PM).
   * The expression DS1302\_Time[3] /10\*16 + DS1302\_Time[3]%10 converts the hour from decimal to BCD format.
   * The DS1302\_WriteByte(DS1302\_HOUR, ...) function writes this BCD value to the hour register of the DS1302.

c

Copy code

DS1302\_WriteByte(DS1302\_MINUTE, DS1302\_Time[4] /10\*16 + DS1302\_Time[4]%10);

1. **Purpose**: This line sets the minute on the DS1302.
2. **How it works**:
   * DS1302\_Time[4] contains the minute value (e.g., 30 for 30 minutes past the hour).
   * The expression DS1302\_Time[4] /10\*16 + DS1302\_Time[4]%10 converts the minute from decimal to BCD format.
   * The DS1302\_WriteByte(DS1302\_MINUTE, ...) function writes this BCD value to the minute register of the DS1302.

c

Copy code

DS1302\_WriteByte(DS1302\_SECOND, DS1302\_Time[5] /10\*16 + DS1302\_Time[5]%10);

1. **Purpose**: This line sets the second on the DS1302.
2. **How it works**:
   * DS1302\_Time[5] contains the second value (e.g., 45 for 45 seconds).
   * The expression DS1302\_Time[5] /10\*16 + DS1302\_Time[5]%10 converts the second from decimal to BCD format.
   * The DS1302\_WriteByte(DS1302\_SECOND, ...) function writes this BCD value to the second register of the DS1302.

c

Copy code

DS1302\_WriteByte(DS1302\_DAY, DS1302\_Time[6] /10\*16 + DS1302\_Time[6]%10);

1. **Purpose**: This line sets the day of the week on the DS1302.
2. **How it works**:
   * DS1302\_Time[6] contains the day of the week value (e.g., 03 for Wednesday).
   * The expression DS1302\_Time[6] /10\*16 + DS1302\_Time[6]%10 converts the day of the week from decimal to BCD format.
   * The DS1302\_WriteByte(DS1302\_DAY, ...) function writes this BCD value to the day register of the DS1302.

c

Copy code

DS1302\_WriteByte(DS1302\_WP, 0x80);

}

1. **Purpose**: The final line re-enables write protection on the DS1302.
2. **How it works**:
   * The DS1302\_WriteByte(DS1302\_WP, 0x80); call writes 0x80 to the WP register, re-enabling write protection. This prevents any further accidental modifications to the time and date registers until write protection is disabled again.

### Execution Flow

When DS1302\_SetTime(); is called in the main function, it executes the steps outlined above in sequence. This ensures that the time and date values are correctly set in the DS1302 RTC, allowing the RTC to maintain the current time and date.

* The function first disables write protection.
* Then, it writes each time component (year, month, date, hour, minute, second, and day) in BCD format to the corresponding DS1302 registers.
* Finally, it re-enables write protection to secure the time settings.

This process ensures that the DS1302 RTC is accurately initialized with the desired time and date.

|  |
| --- |
| void **DS1302\_WriteByte**(unsigned char Command, unsigned char Data) |
| { |
| unsigned int i; |
|  |
| // Enable communication with the DS1302 |
| DS1302\_CE = 1; |
|  |
| // Send the Command byte |
| for (i = 0; i < 8; i++) { |
| DS1302\_IO = Command & (0x01 << i); // Set IO pin to the current bit of Command |
| DS1302\_SCLK = 1; // Pulse the clock signal |
| DS1302\_SCLK = 0; |
| } |
|  |
| // Send the Data byte |
| for (i = 0; i < 8; i++) { |
| DS1302\_IO = Data & (0x01 << i); // Set IO pin to the current bit of Data |
| DS1302\_SCLK = 1; // Pulse the clock signal |
| DS1302\_SCLK = 0; |
| } |
|  |
| // Disable communication with the DS1302 |
| DS1302\_CE = 0; |
| } |

**How the DS1302\_WriteByte function works**. This function is used to send a command or data byte to the DS1302 Real-Time Clock (RTC) module. We'll go through the function step-by-step.

### DS1302\_WriteByte Function:

Here's the function again for reference:

c

Copy code

void DS1302\_WriteByte(unsigned char Command, unsigned char Data)

{

unsigned int i;

// Enable communication with the DS1302

DS1302\_CE = 1;

// Send the Command byte

for (i = 0; i < 8; i++) {

DS1302\_IO = Command & (0x01 << i); // Set IO pin to the current bit of Command

DS1302\_SCLK = 1; // Pulse the clock signal

DS1302\_SCLK = 0;

}

// Send the Data byte

for (i = 0; i < 8; i++) {

DS1302\_IO = Data & (0x01 << i); // Set IO pin to the current bit of Data

DS1302\_SCLK = 1; // Pulse the clock signal

DS1302\_SCLK = 0;

}

// Disable communication with the DS1302

DS1302\_CE = 0;

}

### Detailed Explanation

#### 1. **Function Definition**

c

Copy code

void DS1302\_WriteByte(unsigned char Command, unsigned char Data)

* **void**: The function does not return a value.
* **unsigned char Command**: This is the command byte to be sent to the DS1302.
* **unsigned char Data**: This is the data byte to be sent to the DS1302.

#### 2. **Enabling Communication**

c

Copy code

DS1302\_CE = 1;

* **What It Does**: Sets the Chip Enable (CE) pin high.
* **How It Works**: This activates the DS1302 module, signaling that it should start receiving data.

#### 3. **Sending the Command Byte**

c

Copy code

for (i = 0; i < 8; i++) {

DS1302\_IO = Command & (0x01 << i);

DS1302\_SCLK = 1; // Pulse the clock signal

DS1302\_SCLK = 0;

}

* **Purpose**: This loop sends each bit of the command byte to the DS1302.
* **How It Works**:
  + **Command & (0x01 << i)**: This expression extracts the i-th bit from the Command byte.
    - **0x01**: This is a binary 00000001.
    - **0x01 << i**: This shifts the 1 to the i-th bit position. For example, if i is 3, 0x01 << 3 results in 00001000.
    - **Command & (0x01 << i)**: This masks the Command byte to keep only the bit of interest. The result is either 0 or a non-zero value.
  + **DS1302\_IO = Command & (0x01 << i);**: Sets the IO pin to either high or low based on the bit.
  + **DS1302\_SCLK = 1; and DS1302\_SCLK = 0;**: Pulses the clock pin to signal the DS1302 that a bit is being sent. The clock pin goes high to indicate that data is valid, then goes low to prepare for the next bit.

#### 4. **Sending the Data Byte**

c

Copy code

for (i = 0; i < 8; i++) {

DS1302\_IO = Data & (0x01 << i);

DS1302\_SCLK = 1; // Pulse the clock signal

DS1302\_SCLK = 0;

}

* **Purpose**: Similar to the command byte, this loop sends each bit of the data byte to the DS1302.
* **How It Works**:
  + **Data & (0x01 << i)**: Extracts the i-th bit from the Data byte, similar to how it was done for the command byte.
  + **DS1302\_IO = Data & (0x01 << i);**: Sets the IO pin to the bit's value.
  + **DS1302\_SCLK = 1; and DS1302\_SCLK = 0;**: Pulses the clock pin to signal the DS1302 to read the bit.

#### 5. **Disabling Communication**

c

Copy code

DS1302\_CE = 0;

* **What It Does**: Sets the Chip Enable (CE) pin low.
* **How It Works**: This deactivates the DS1302 module, ending the communication session.

### Summary

1. **Enable Communication**: Sets the CE pin high to activate the DS1302.
2. **Send Command Byte**: Sends each bit of the command byte, pulsing the clock pin to signal each bit.
3. **Send Data Byte**: Sends each bit of the data byte in the same way.
4. **Disable Communication**: Sets the CE pin low to end the communication.

Each part of this function works together to ensure the DS1302 receives the correct command and data. The clock pulses are crucial for the DS1302 to correctly read each bit of the command and data bytes.

|  |
| --- |
| unsigned char DS1302\_ReadByte(unsigned char Command) |
| { |
| unsigned char i, Data = 0x00; |
| Command |= 0x01; |
| DS1302\_CE = 1; |
|  |
| for (i = 0; i < 8; i++) |
| { |
| DS1302\_IO = Command & (0x01 << i); |
| DS1302\_SCLK = 0; |
| DS1302\_SCLK = 1; |
| } |
|  |
| DS1302\_IO = 1; // Set to input mode |
|  |
| for (i = 0; i < 8; i++) |
| { |
| DS1302\_SCLK = 1; |
| DS1302\_SCLK = 0; |
|  |
| if (DS1302\_IO) |
| { |
| Data |= (0x01 << i); |
| } |
| } |
|  |
| DS1302\_CE = 0; |
| DS1302\_IO = 0; // Set IO to low as default |
|  |
| return Data; |
| } |

**Break down the DS1302\_ReadByte function** line by line to understand how it works. This function is used to read a byte of data from the DS1302 Real-Time Clock (RTC) module.

### DS1302\_ReadByte Function:

Here is the function for reference:

c

Copy code

unsigned char DS1302\_ReadByte(unsigned char Command)

{

unsigned char i, Data = 0x00;

Command |= 0x01;

DS1302\_CE = 1;

for (i = 0; i < 8; i++)

{

DS1302\_IO = Command & (0x01 << i);

DS1302\_SCLK = 0;

DS1302\_SCLK = 1;

}

DS1302\_IO = 1; // Set to input mode

for (i = 0; i < 8; i++)

{

DS1302\_SCLK = 1;

DS1302\_SCLK = 0;

if (DS1302\_IO)

{

Data |= (0x01 << i);

}

}

DS1302\_CE = 0;

DS1302\_IO = 0; // Set IO to low as default

return Data;

}

### Detailed Explanation

#### 1. **Function Definition**

c

Copy code

unsigned char DS1302\_ReadByte(unsigned char Command)

* **unsigned char**: The function returns an 8-bit unsigned value (a byte).
* **Command**: This is the command byte to tell the DS1302 which data to read.

#### 2. **Modify Command for Read Operation**

c

Copy code

Command |= 0x01;

* **What It Does**: This sets the least significant bit (LSB) of the Command byte to 1.
* **Why**: The DS1302 uses this bit to determine if the operation is a read operation (when this bit is set to 1).

#### 3. **Enable Communication**

c

Copy code

DS1302\_CE = 1;

* **What It Does**: Sets the Chip Enable (CE) pin high.
* **Why**: This activates the DS1302, preparing it to receive the command and send data.

#### 4. **Send the Command Byte**

c

Copy code

for (i = 0; i < 8; i++)

{

DS1302\_IO = Command & (0x01 << i);

DS1302\_SCLK = 0;

DS1302\_SCLK = 1;

}

* **Purpose**: This loop sends each bit of the command byte to the DS1302.
* **How It Works**:
  + **Command & (0x01 << i)**: This extracts the i-th bit of the Command.
    - **0x01**: Binary 00000001.
    - **0x01 << i**: Shifts 1 to the i-th bit position.
    - **Command & (0x01 << i)**: Masks the Command byte to isolate the bit to be sent.
  + **DS1302\_IO = Command & (0x01 << i);**: Sets the IO pin to either high or low based on the bit.
  + **DS1302\_SCLK = 0; and DS1302\_SCLK = 1;**: Pulses the clock pin to tell the DS1302 that a bit has been sent.

#### 5. **Switch IO to Input Mode**

c

Copy code

DS1302\_IO = 1;

* **What It Does**: Sets the IO pin to input mode.
* **Why**: Now the DS1302 will send data to the microcontroller, so the IO pin needs to be set to receive mode.

#### 6. **Read the Data Byte**

c

Copy code

for (i = 0; i < 8; i++)

{

DS1302\_SCLK = 1;

DS1302\_SCLK = 0;

if (DS1302\_IO)

{

Data |= (0x01 << i);

}

}

* **Purpose**: This loop reads each bit of the data byte from the DS1302.
* **How It Works**:
  + **DS1302\_SCLK = 1; and DS1302\_SCLK = 0;**: Pulses the clock pin to signal that the DS1302 should send the next bit.
  + **if (DS1302\_IO)**: Checks the state of the IO pin to determine if the bit is high or low.
    - **Data |= (0x01 << i);**: Sets the corresponding bit in the Data variable if the IO pin is high.

#### 7. **Disable Communication**

c

Copy code

DS1302\_CE = 0;

* **What It Does**: Sets the Chip Enable (CE) pin low.
* **Why**: This ends the communication session with the DS1302.

#### 8. **Reset IO Pin**

c

Copy code

DS1302\_IO = 0; // Set IO to low as default

* **What It Does**: Sets the IO pin to low.
* **Why**: This ensures that the IO pin is in a known state (low) after the operation.

#### 9. **Return the Data**

c

Copy code

return Data;

* **What It Does**: Returns the byte of data read from the DS1302.
* **Why**: This allows the calling function to use the data that was read.

### Summary

1. **Modify Command**: Set the command to read mode by adding 1 to it.
2. **Enable Communication**: Activate the DS1302 module by setting the CE pin high.
3. **Send Command**: Send the command byte bit by bit to the DS1302.
4. **Switch to Input**: Change the IO pin to input mode to receive data.
5. **Read Data**: Read the data byte bit by bit from the DS1302.
6. **Disable Communication**: Deactivate the DS1302 module by setting the CE pin low.
7. **Reset IO Pin**: Set the IO pin to low to ensure a known state.
8. **Return Data**: Return the data read from the DS1302 to the caller.

Each step ensures that the DS1302 properly receives the command and sends back the requested data.