Project Report: PIC18F4550 & DS3232 Real-Time Clock

Author: Student
Date: 10/24/2025
Project: Digital RTC Clock with Set Function

1. Project Overview

Objective

The goal of this project is to build a reliable, battery-backed digital clock. The system uses a **PIC18F4550 microcontroller** as the "brain" to read time and date from a **DS3232 Real-Time Clock (RTC)** "watch" chip. The time and date are then displayed on a **16x2 LCD screen**. The user can also set the time and date using two push-buttons.

Key Features

- **Real-Time Display:** Shows hours, minutes, seconds, date, month, and year.
- **Time Setting:** An "Edit" button and an "Increment" button allow the user to set all time/date parameters.
- **Non-Volatile Timekeeping:** The DS3232 chip has its own battery (or a V_{BAT} pin), so it keeps time accurately even if the main microcontroller is powered off.
- **Software I2C:** The PIC microcontroller communicates with the RTC chip using a software-based I2C protocol (bit-banging).

2. Hardware Explained (In Simple Words)

This is what each electronic part in your schematic does.

- **U1 (PIC18F4550 Microcontroller):**
  - **What it is:** This is the **"brain"** of the project.
  - **What it does:** It runs your C code. Its job is to tell the LCD screen what to write, read the buttons, and constantly ask the DS3232 chip, "What time is it?"
- **U2 (DS3232 Real-Time Clock):**
  - **What it is:** This is the **"watch."**
  - **What it does:** This chip's only job is to keep perfect time. It has its own internal crystal and battery backup. It doesn't care if the PIC is on or off; it just keeps counting the seconds, minutes, hours, and date. The PIC microcontroller must *ask* this chip for the time.
- **LCD1 (LM016L Display):**
  - **What it is:** This is the **"screen."**
  - **What it does:** It's a 16x2 Liquid Crystal Display, meaning it shows 2 lines of 16 characters each. It's a "dumb" screen; it only shows the exact characters that the

"brain" (PIC) tells it to display.
- **Edit/Inc Buttons (Push Buttons):**
  - **What they are:** The **"controls."**
  - **What they do:** These are simple buttons that connect to the PIC. The code constantly checks if they are pressed.
    - BTN_EDIT tells the code to enter "setting mode."
    - BTN_INC is used to increase the numbers (like changing the hour from 10 to 11).
- **R1, R2, R3, R4 (10k Resistors):**
  - **What they are: "Pull-Up" Resistors.**
  - **What they do:** These resistors are essential. They keep the data lines (SDA, SCL) and the button lines in a "high" (or 1) state by default.
    - **For Buttons:** When you *press* a button, you connect the line to Ground (0), so the PIC sees the signal change from 1 to 0. Without these, the PIC wouldn't know if the button was pressed or not.
    - **For I2C:** The I2C protocol *requires* these resistors to work.
- **VEE Pin & Potentiometer (Not in diagram, but required):**
  - **What it is:** The **"contrast knob."**
  - **What it does:** The VEE pin on the LCD controls how dark the characters are. Connecting it directly to ground (as in the original image) makes the screen blank in Proteus. You must use a potentiometer (variable resistor) to "dial in" the voltage and make the text visible.

3. Software Explained (What Each Function Does)

This is a simple breakdown of your main.c file, function by function.

Main Program

- **main()**
  - This is the **starting point** of your entire program.
  - It does two things:
    1. **Setup (runs once):** Configures the PIC's clock speed, sets up the ports (PORTD for LCD, PORTB for buttons/I2C), and calls LCD_Init() and I2C_Init().
    2. **Main Loop (runs forever):** A while(1) loop that:
       - Checks if the BTN_EDIT button is pressed. If yes, it calls the edit() function to set the time.
       - If not editing, it reads the current time from the DS3232.
       - Calls DS3232_Display() to show the time on the LCD.
       - Waits 50ms and does it all over again.

LCD Driver Functions (Controls the Screen)

- **LCD_Init()**
  - **"Wakes up the screen."**
  - Sends a very specific sequence of commands to the LCD to tell it, "We are going to

talk in 4-bit mode, you have 2 lines, and you're turned on." This is the part that was fixed.

- **LCD_Command(uint8_t cmd)**
  - **"Tells the screen WHAT TO DO."**
  - Sends a command byte to the LCD. Examples: 0x01 (Clear screen), 0x80 (Move cursor to line 1).
- **LCD_Data(uint8_t data)**
  - **"Tells the screen WHAT TO WRITE."**
  - Sends a data byte to the LCD. This is how you print a single character, like 'A' or '5', to the screen.
- **LCD_String(const char *str)**
  - **"Writes a whole sentence."**
  - A helper function that calls LCD_Data() repeatedly to print a full string of text (like "TIME: ").
- **LCD_Clear()**
  - **"Wipes the screen clean."**
  - A shortcut for LCD_Command(0x01).
- **LCD_SetCursor(uint8_t col, uint8_t row)**
  - **"Moves the typing position."**
  - Tells the LCD where the *next* character you send should appear (e.g., "move to row 2, column 1").

I2C Driver Functions (Talks to the Clock)

- **I2C_Init()**
  - **"Prepares the communication line."**
  - Sets the two I2C pins (SDA and SCL) as inputs, making them high (the default "idle" state).
- **I2C_Start() / I2C_Stop()**
  - **"Starts/Stops the conversation."**
  - These send the special electrical signals that tell the DS3232 chip: "Hey, listen!" (Start) and "Okay, I'm done talking." (Stop).
- **I2C_Write(uint8_t data)**
  - **"Sends 1 byte of data"** (8 bits).
  - This is used to send data *to* the clock. For example: "I want to talk to device 0xD0" or "I'm setting the new 'minutes' value to 0x30."
- **I2C_Read(uint8_t ack)**
  - **"Receives 1 byte of data."**
  - This is used to get data *from* the clock. For example: "Please tell me the current 'seconds' value."
- **I2C_Delay()**
  - **"A tiny pause."**
  - Because this is "Software I2C," we have to create our own clock speed. This tiny

delay ensures the speed is slow enough for the chips to understand (about 100kHz).

Clock Logic Functions

- **DS3232_Display()**
  - **"Formats and shows the time."**
  - The DS3232 stores time in a weird format called **BCD** (Binary Coded Decimal). For example, the number 23 is stored as 0x23.
  - This function's job is to:
    1. Convert the BCD value (0x23) into a normal decimal number (23).
    2. Convert that number into text characters ('2' and '3').
    3. Put those characters into the time[] and calendar[] strings.
    4. Finally, print those strings to the LCD.
- **blink()**
  - **"A flashing delay."**
  - This is a special __delay_ms(250) that constantly checks if a button is being pressed. This allows the edit() function to flash a number on and off while still being responsive to button presses.
- **edit(uint8_t parameter, uint8_t x, uint8_t y)**
  - **"The time-setting logic."**
  - This is the heart of the "set time" feature. When called, it:
    1. Flashes the number you are editing (like the hour) at position (x, y).
    2. Waits for you to press BTN_INC.
    3. When BTN_INC is pressed, it increases the number, making sure it stays in-bounds (e.g., hours wrap from 23 back to 0).
    4. Waits for you to press BTN_EDIT.
    5. When BTN_EDIT is pressed, it stops flashing, saves the new number, and returns it.

4. How It Works: Operational Flow

1. **Power On:** The PIC18F4550 starts. It runs LCD_Init() and I2C_Init().
2. **Main Loop:** The PIC enters its while(1) loop.
3. **Read Time:** It sends a "start" signal, the DS3232's address (0xD0), and the register to start reading from (0x00 for seconds).
4. **Receive Time:** It performs a "repeated start" and reads 7 bytes of data (Sec, Min, Hr, Day, Date, Mo, Yr) from the DS3232. It sends a "stop" signal.
5. **Display Time:** The DS3232_Display() function converts these 7 bytes from BCD to ASCII text and updates the LCD.
6. **Check for Edit:** The code checks if BTN_EDIT is pressed.
   - **If NO:** It waits 50ms and goes back to step 2.
   - **If YES:** It enters the "Edit" sequence.
7. **Edit Sequence:**
   - It calls edit(hour, ...) to edit the hour. The user presses BTN_INC to change the value

and BTN_EDIT to confirm.
    - It then calls edit(minute, ...) for minutes.
    - ...and so on for Date, Month, and Year.
8. **Write Time:** After all values are edited, the PIC performs an I2C_Write operation, sending all the new BCD-converted values (hour, minute, etc.) back to the DS3232 to save them.
9. **Resume:** The code returns to the main while(1) loop at step 2.