REAL-TIME EMBEDDED SYSTEM
EXPERIMENT NO. 5

Real-Time Clock

Name: NAVARRO, ROD GERYK C.

Course/Section: CPE161P-4/C1

Group No.: N/A

Date of Performance: 01/19/2025
Date of Submission: 02/2025

CYREL O. MANLISES, PH.D.
Instructor

DISCUSSION

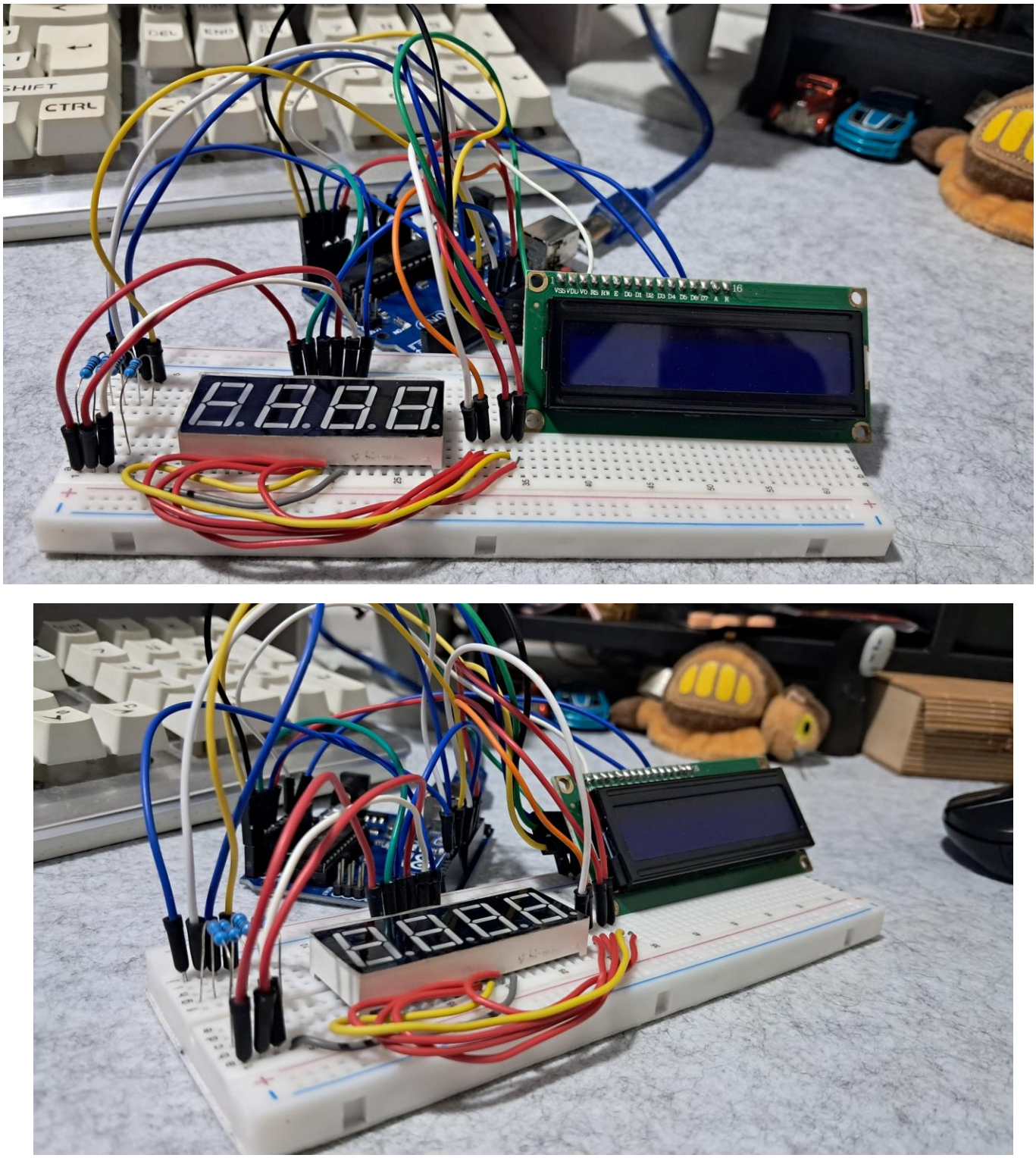


Figure 1: The Connection of the 4-Digit Seven Segment Display, and the LCD

This experiment focuses on building a real-time digital clock using an Arduino UNO. The first step was to assemble the circuit by connecting a 4-digit seven-segment display and an LCD to the Arduino. The seven-segment display is responsible for showing the hours and minutes, where the first two digits from the left represent the hours, and the last two digits represent the minutes. Meanwhile, a 16x2 I2C LCD is used to display the seconds.

The clock operates using the Arduino's millis() function, which keeps track of time. The seconds update every second, and every 60 seconds, the minutes increase by one. The hours follow a 12-hour format, meaning they reset after reaching 12. The LCD updates the seconds in real-time, ensuring accurate timekeeping. Additionally, specific functions are programmed to handle time increments and display updates, making the clock function properly.

Navarro_EXP5_CPE161P.ino

```
#include <LiquidCrystal_I2C.h>
const int sPins[7] = {A3,A2,A1,12,8,9,10};
const int dPins[4] = {4,5,6,7};

int hours = 0;
int minutes = 0;
int seconds = 0;
bool adjustMinutes = true;
bool incrementMode = true;
LiquidCrystal_I2C lcd(0x27, 16,2);

unsigned long previousMillis = 0;
unsigned long previousMillisSeconds = 0;
const long interval = 60000;
const long intervalSeconds = 1000;
const int debounceDelay = 50;

byte digitCodes[10] = {
  B00111111,
  B00000110,
  B01011011,
  B01001111,
  B01100110,
  B01101101,
  B01111101,
  B00000111,
  B01111111,
  B01101111
};
```

Figure 2: *The Initialization of Variables and Components*

In this experiment, I started by defining the necessary variables and pin assignments for the real-time digital clock. The sPins array stores the pin numbers for the seven-segment display, which controls the individual segments to show numbers. The dPins array is used to control which digit is currently active on the display. I also initialized hours, minutes, and seconds as integer variables to keep track of time. Additionally, I included two Boolean variables, adjustMinutes and incrementMode, which can be used to modify the behavior of the clock when needed. The lcd object is created using the LiquidCrystal_I2C library, allowing communication with the 16x2 LCD display. To ensure accurate timekeeping, I also defined previousMillis and previousMillisSeconds, which work with the millis() function to update time at the correct intervals.

Navarro_EXP5_CPE161P.ino

```
31  void setup() {
32    lcd.init();
33    lcd.backlight();
34    for (int i = 0; i < 7; i++){
35      pinMode(sPins[i], OUTPUT);
36    }
37    for(int i = 0; i < 4; i++){
38      pinMode(dPins[i], OUTPUT);
39      digitalWrite(dPins[i], HIGH);
40    }
41
42    displayTime(hours, minutes);
43    displaySeconds(seconds);
44
45  }
```

Figure 3: The Setup of the Components

The setup() function is responsible for initializing the components. First, the LCD is set up with lcd.init() and lcd.backlight() to turn on the display. Then, I configured the seven-segment display by setting all segment pins (sPins) as output. Similarly, the digit control pins (dPins) are set as output and initialized to HIGH, ensuring that all digits remain off until needed. Finally, the displayTime() and displaySeconds() functions are called to show the initial time values on both the seven-segment display and the LCD. This ensures that the system starts with a visible display of the current time.

```
void incrementMinute(){
  minutes++;
  if(minutes >= 60){
    minutes = 0;
    incrementHour();
  }
}
```

```
void displaySeconds(int secs){
  lcd.clear();
  lcd.setCursor(1,1);

  if(secs < 10) lcd.print("0");
  lcd.print(secs);

}
```

Figure 4: Fixing the Errors

During the experiment, I encountered an issue where the clock was not updating the time correctly. After debugging, I realized that the incrementMinute() function was commented out in the loop() function. This prevented the minutes from increasing every 60 seconds. To fix this, I simply uncommented the incrementMinute() function inside the time-checking condition. Another issue was that the LCD did not properly clear previous values when updating the seconds. To solve this, I used lcd.clear() inside the displaySeconds() function, ensuring that the display updates correctly without overlapping digits.

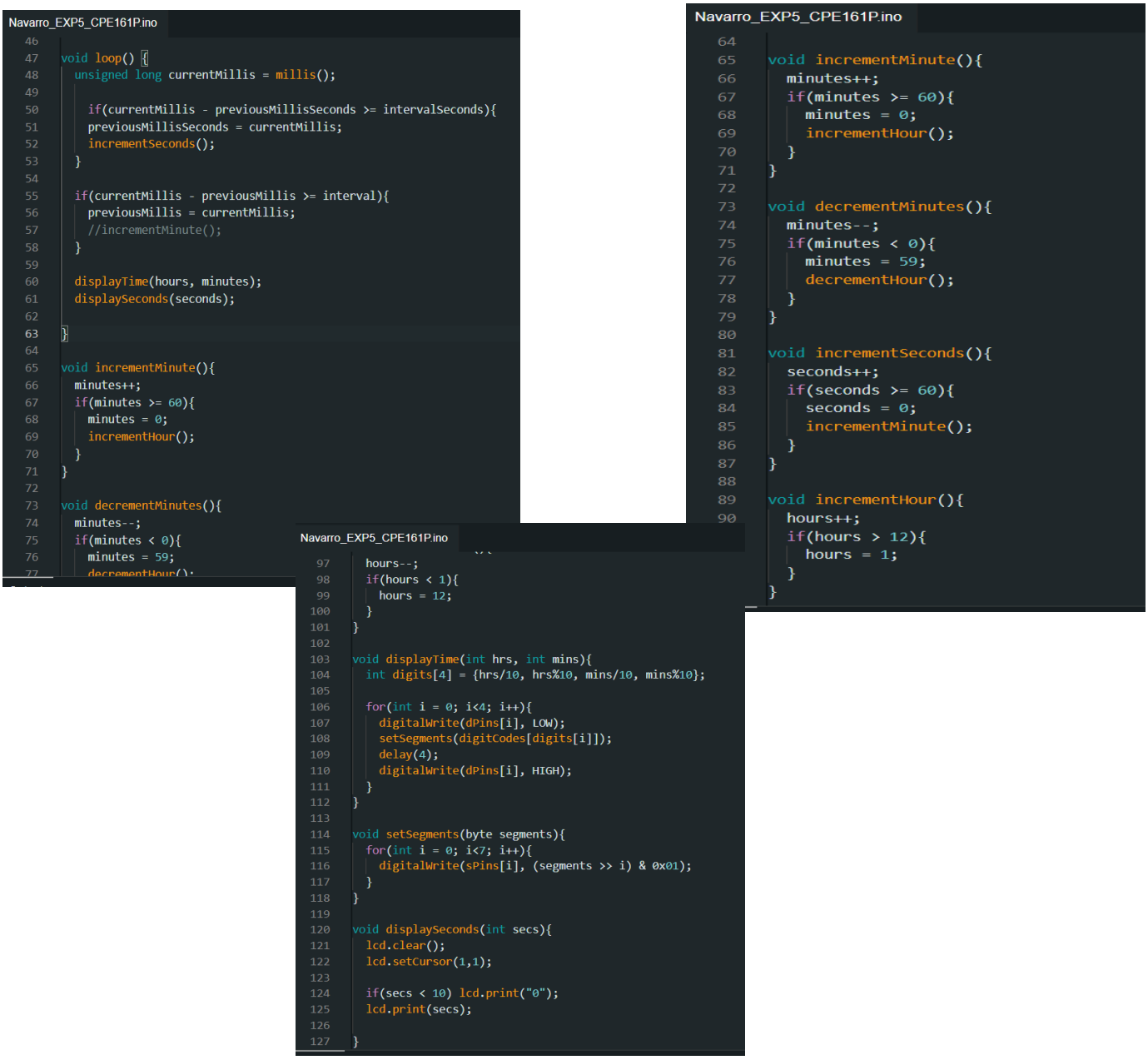
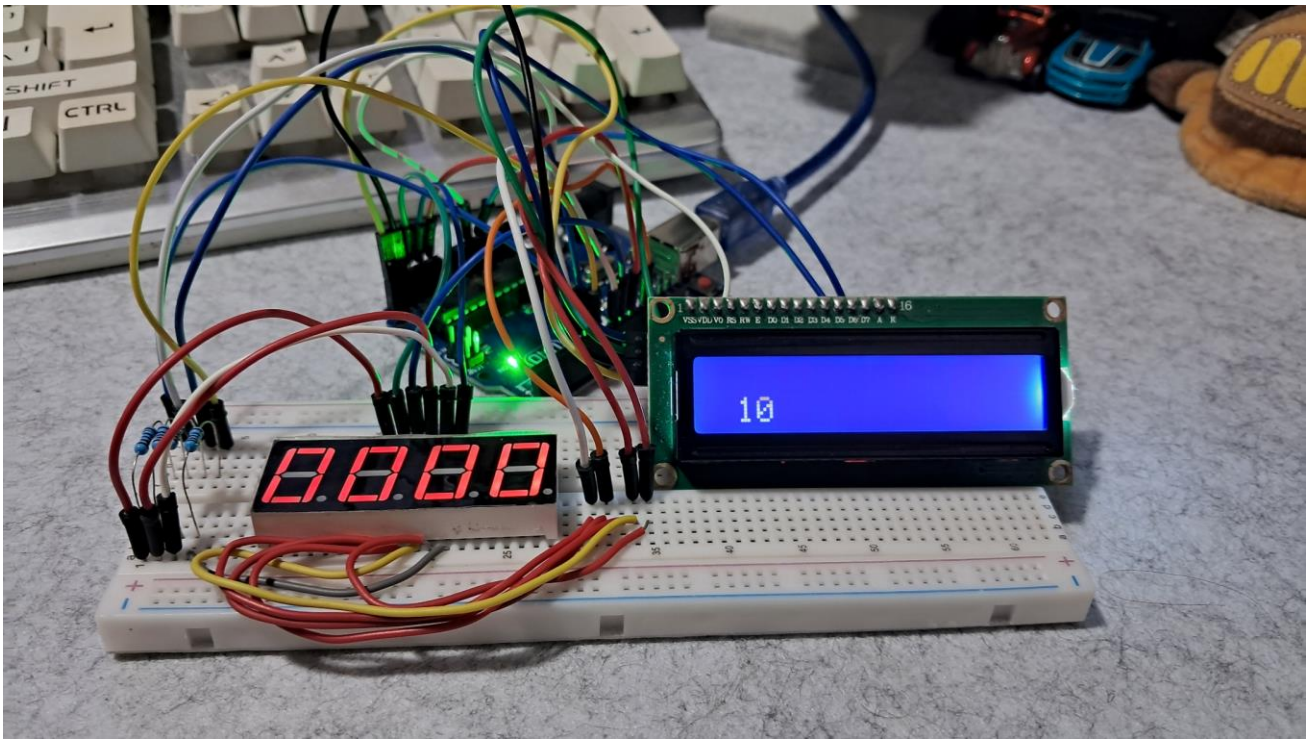
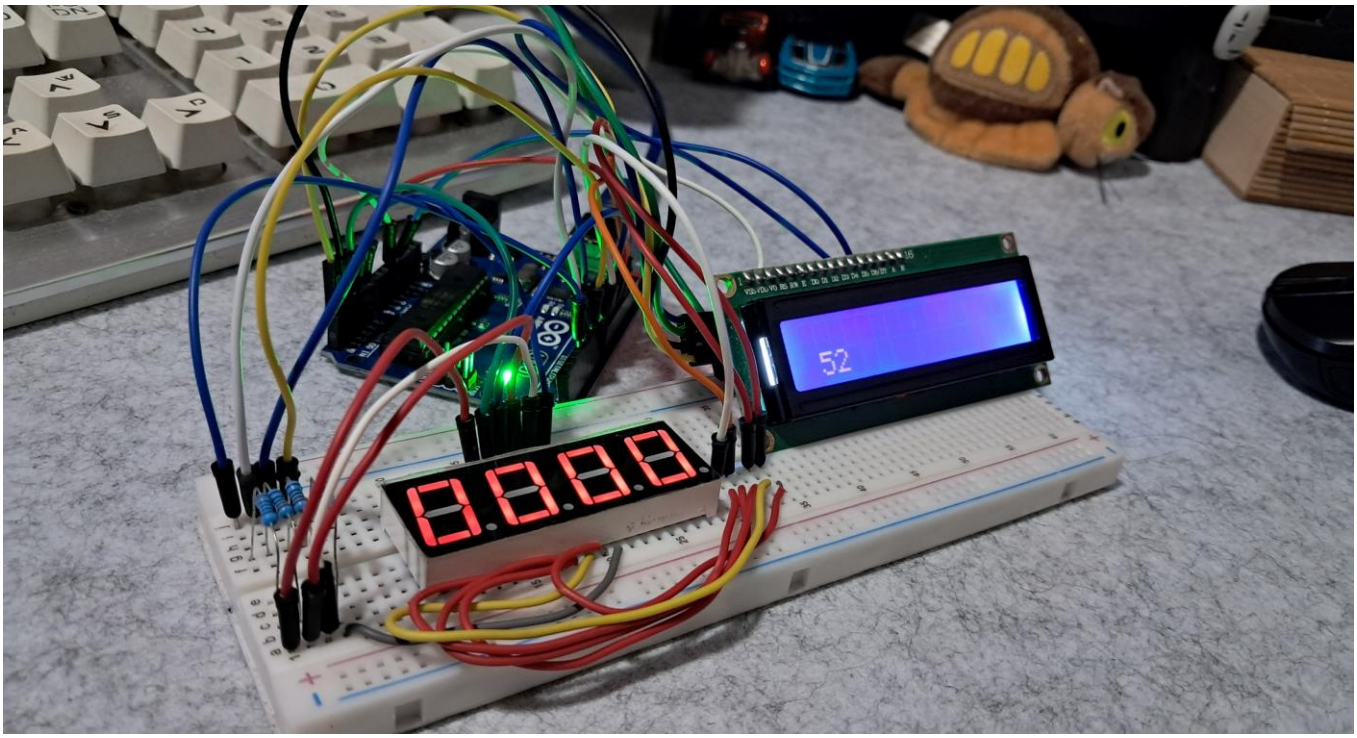


Figure 5: Loop Function and other Necessary Functions

The loop() function continuously updates the time. It first checks if one second has passed using millis(). If so, it calls incrementSeconds() to update the seconds. When the seconds reach 60, the function automatically increments the minutes using incrementMinute(), which in turn updates the hours when needed. The displayTime() function then refreshes the seven-segment display, while displaySeconds() updates the LCD in real time. These functions work together to keep the clock running accurately. Through this experiment, I learned how to use millis() to manage time without using the delay() function, which ensures smooth operation of the clock.


SHIFT
CTRL
10


52

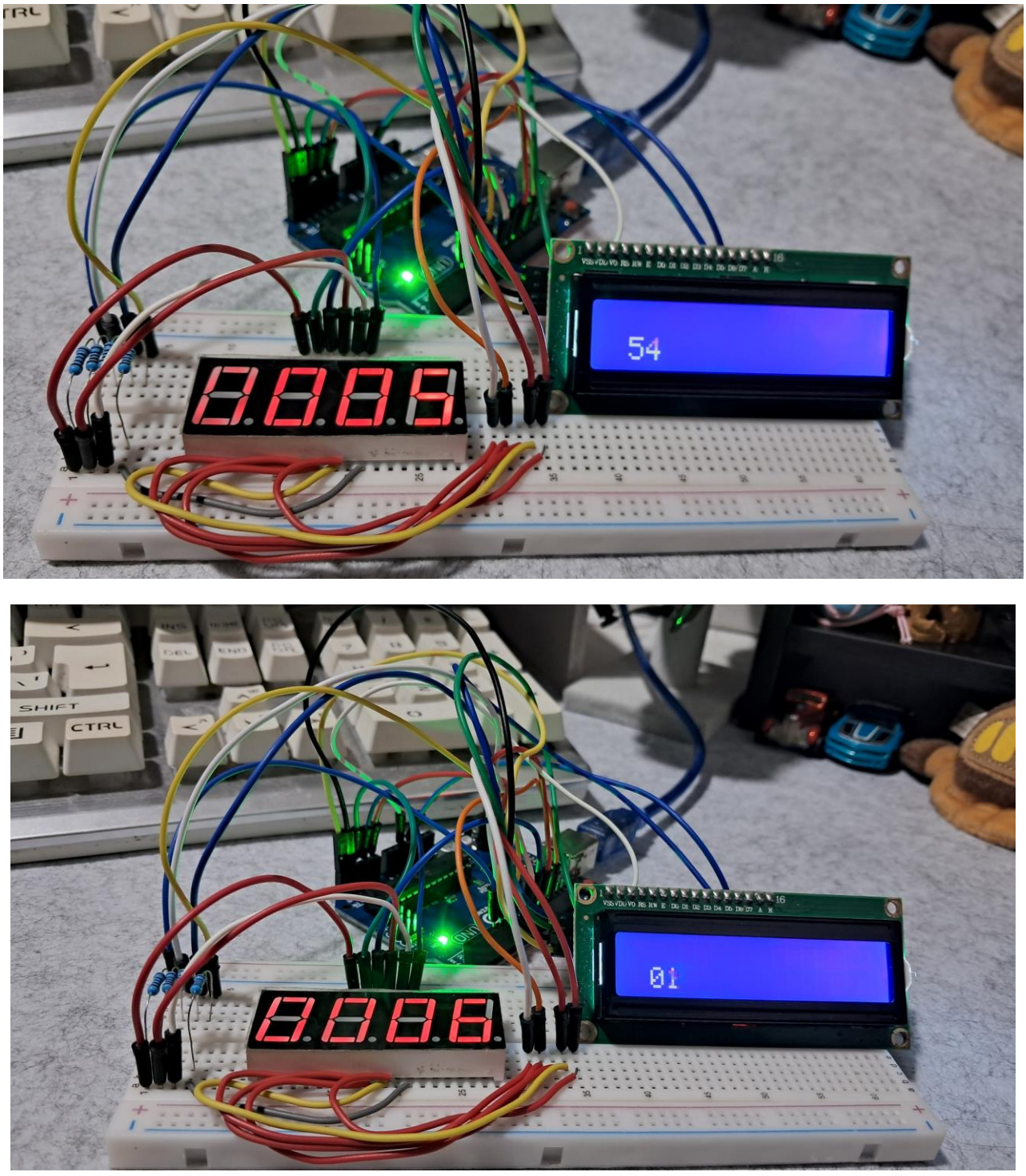


Figure 6: Working Prototype.

This is the final working prototype of the real-time clock experiment, which successfully meets all the requirements.

CONCLUSION

In this experiment, I successfully built a real-time digital clock using an Arduino UNO, a 4-digit seven-segment display, and an LCD. The system accurately tracks time using the millis() function, allowing the seconds, minutes, and hours to update smoothly. The seven-segment display shows the hours and minutes, while the LCD displays the seconds in real time. I carefully programmed functions to manage time increments and display updates, ensuring that the clock runs correctly. Through troubleshooting, I also identified and fixed issues, such as the minutes not updating due to a commented-out function and overlapping digits on the LCD, which I resolved using lcd.clear().

This experiment helped me understand how to create a functional digital clock without using the delay() function, making the system more efficient. I learned the importance of using millis() for timekeeping and properly managing display updates. Setting up the seven-segment display and LCD also improved my knowledge of how to control multiple components simultaneously. This experience gave me valuable skills in programming and debugging, which will be useful for future Arduino projects.