2023

Design Report

CNSCC.369: Embedded System

肖云轩 19726069

Content

Lab 1

Task 1:

Blinking LEDs Write a block of code to turn all the PORTC LEDs on and offs

Lab Report:

Problem Statement:

The task is to write a code to blink all the LEDs on PORTC. The code should turn all the LEDs on and then off after 1 second delay, and then repeat the process indefinitely.

Solution:

The given code achieves the desired functionality of blinking all the LEDs on PORTC. The code initializes the PORTC as output and then enters into an endless loop. Inside the loop, it first turns off all the LEDs on PORTA, PORTB, PORTC, PORTD, and PORTE by setting their corresponding LAT (Latch) registers to 0x00. It then adds a delay of 1 second using the Delay_ms() function.

Next, the code turns on all the LEDs on the same ports by setting their corresponding LAT registers to 0xFF. It then adds another delay of 1 second. Finally, the loop repeats, and the process continues indefinitely.

In conclusion, the provided code successfully blinks all the LEDs on PORTC and can be used as a simple way to verify the functionality of the microcontroller's output ports.

Code

```
void main() {
  TRISC = 0;              // set direction to be output
  do {
    LATA = 0x00;          // Turn OFF LEDs on PORTA
    LATB = 0x00;          // Turn OFF LEDs on PORTB
    LATC = 0x00;          // Turn OFF LEDs on PORTC
    LATD = 0x00;          // Turn OFF LEDs on PORTD
    LATE = 0x00;          // Turn OFF LEDs on PORTE
    Delay_ms(1000);       // 1 second delay

    LATA = 0xFF;          // Turn ON LEDs on PORTA
    LATB = 0xFF;          // Turn ON LEDs on PORTB
    LATC = 0xFF;          // Turn ON LEDs on PORTC
    LATD = 0xFF;          // Turn ON LEDs on PORTD
    LATE = 0xFF;          // Turn ON LEDs on PORTE
    Delay_ms(1000);       // 1 second delay
  } while(1);             // Endless loop
}
```

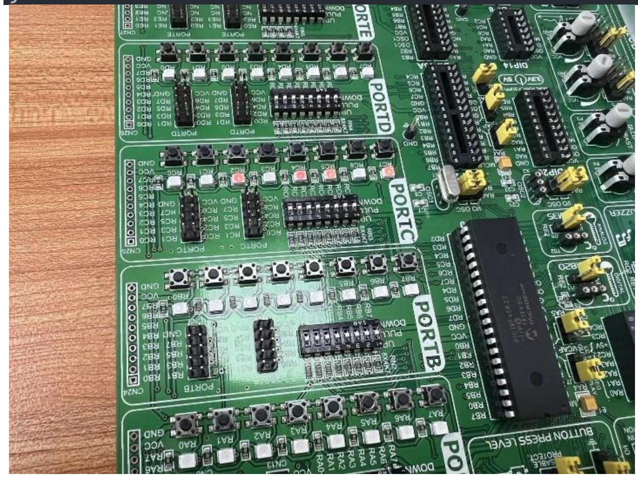
Task 2:

Problem Statement:
Write a block of code to turn on the PORTC LEDs according to a sequence of random integer numbers. Each random integer number is generated between 1 and 255, and the LEDs are turned on to indicate this number in binary. The LEDs are turned off after a delay of 1 second.

Solution:
To solve this problem, we can use the rand() function to generate a random integer between 1 and 255. Then, we can convert this integer to binary and turn on the corresponding PORTC LEDs to represent the binary number.

We can use the given function "convertDecimaltoBinary" to convert the decimal number to binary. This function takes an integer as input and returns its binary equivalent.

```
// This function takes a decimal number as input and converts it to a binary
number.
int covertDecimaltoBinary(int n){
    int binaryNumber = 0;
    int remainder, i = 1;
    while(n != 0){
    remainder = n % 2; // Find the remainder when dividing n by 2
    n /= 2; // Divide n by 2 and store the result
    binaryNumber += remainder*i; // Multiply the remainder by i and add it to
the binary number
    i *= 10; // Multiply i by 10 to move to the next place value in the
binary number
    }
    return binaryNumber; // Return the binary number
}

void main() {

    TRISC = 0; // Set the direction of PORTC to be output

    do {
    int randomNumber = rand()%256; // Generate a random integer between 1 and
255
    LATA = 0x00; // Turn OFF LEDs on PORTA
    LATB = 0x00; // Turn OFF LEDs on PORTB
    LATC = covertDecimaltoBinary(randomNumber); // Convert the random integer
to binary and turn on the corresponding LEDs on PORTC
    LATD = 0x00; // Turn OFF LEDs on PORTD
    LATE = 0x00; // Turn OFF LEDs on PORTE
    Delay_ms(1000); // Delay for 1 second

    } while(1); // Endless loop
}
```



Task 3:

Write a block of code to turn on the PORTC LEDs according to the state of the PORTB switches. The first LED should reflect the state of the first switch, and the second LED should reflect the state of the second switch, and so on. An LED is turned off when the corresponding switch is pressed, and is turned on otherwise. Make sure your code works when multiple switches are pressed together.

Lab Report: Turn on PORTC LEDs according to the state of the PORTB switches

Problem Statement:

Write a block of code to turn on the PORTC LEDs according to the state of the PORTB switches. The first LED should reflect the state of the first switch, and the second LED should reflect the state of the second switch, and so on. An LED is turned off when the corresponding switch is pressed, and is turned on otherwise. Make sure your code works when multiple switches are pressed together.

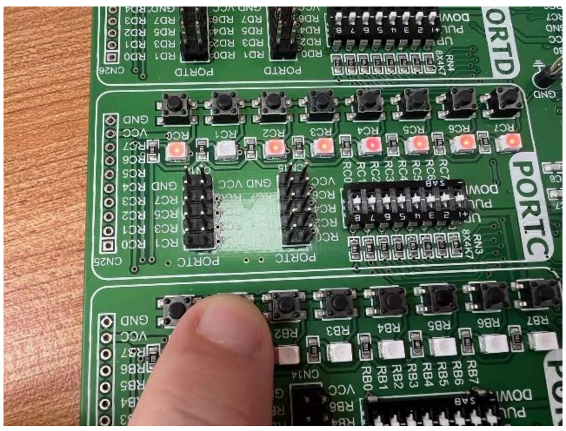
Solution:

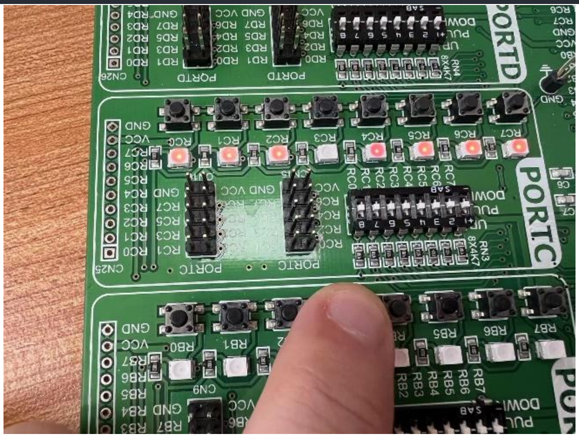
The following registers need to be used in the code to control the hardware:

TRISA: Used to set the direction of the pins on PORTA. 0 for output, 1 for input.
TRISB: Used to set the direction of the pins on PORTB. 0 for output, 1 for input.
TRISC: Used to set the direction of the pins on PORTC. 0 for output, 1 for input.
TRISD: Used to set the direction of the pins on PORTD. 0 for output, 1 for input.
TRISE: Used to set the direction of the pins on PORTE. 0 for output, 1 for input.
PORTC: Used to control the output level of PORTC.

Based on the problem statement, we need to set PORTB as input and PORTC as output, and then control the state of the PORTC LEDs based on the state of the PORTB switches. The code is as follows:

```
void main() {

    TRISA = 0; // set direction of PORTA as output
    TRISB = 0xFF; // set direction of PORTB as input
    TRISC = 0x00; // set direction of PORTC as output
    TRISD = 0; // set direction of PORTD as output
    TRISE = 0; // set direction of PORTE as output
    PORTC = 0xff; // turn on all LEDs connected to PORTC

    do {
        PORTC = ~PORTB; // toggle the LEDs connected to PORTC based on the
value of PORTB
        } while(1); // Endless loop
}
```





Task 4:

Dice via 7-Segment Display Write a block of code to turn on the first digit of the seven segment display according to a switchcontrolled dice. A random dice number is generated between 0 and 9 when a PORTB switch is pressed, and the seven segment digit is turned on to indicate this number. The digit is turned off when the switch is pressed again.

Lab Report:

Problem Statement:

Write a code block to control a seven-segment display to show the number generated by a switch-controlled dice. The code should generate a random number between 0 and 9 when a switch on PORTB is pressed and turn on the corresponding digit on the seven-segment display to indicate this number. The digit should be turned off when the switch is pressed again.

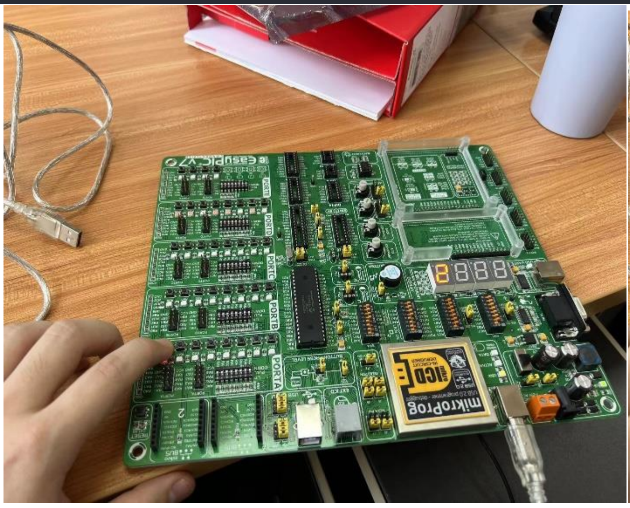
Solution:

The given code defines a function getDigitCode() to return the seven-segment display code for a given number. It also sets the required registers for controlling the hardware.

To implement the solution, we need to set the PORTD register as output and PORTA register as input. We also need to check if the switch on PORTB is pressed. If it is pressed, we generate a random number between 0 and 9 and get the corresponding seven-segment display code using the getDigitCode() function. We then display this code on the PORTD register to turn on the corresponding digit on the seven-segment display. We also turn off the display when the switch is pressed again.

The modified code is as follows:

```
// Function to get the 7-segment display code for a given number
char getDigitCode(int number) {
    char digitCode;

    switch(number) {
        case 0:
            digitCode = 0b00111111;   // Display 0
            break;
        case 1:
            digitCode = 0b00000110;   // Display 1
            break;
        case 2:
            digitCode = 0b01011011;   // Display 2
            break;
        case 3:
            digitCode = 0b01001111;   // Display 3
            break;
        case 4:
            digitCode = 0b01100110;   // Display 4
            break;
        case 5:
            digitCode = 0b01101101;   // Display 5
            break;
        case 6:
            digitCode = 0b01111101;   // Display 6
            break;
        case 7:
            digitCode = 0b00000111;   // Display 7
            break;
        case 8:
            digitCode = 0b01111111;   // Display 8
            break;
        case 9:
            digitCode = 0b01101111;   // Display 9
            break;
        default:
            digitCode = 0x00;   // Display blank
            break;
    }

    return digitCode;
}

// Main function
void main() {
  ANSELD = 0;   // Set PORTD to digital mode
  TRISD = 0;     // Set PORTD as output
  TRISA = 0xFF; // Set PORTA as input
  unsigned char pattern = 0;
  unsigned char cnt = 0;
  for(;;) {
     // Check if switch on PORTB is pressed
     if(PORTB.F0 == 1) {
         pattern = getDigitCode(rand() % 10); // Generate random number
between 0 and 9
         PORTD = pattern; // Display the corresponding digit on 7-segment
display
         Delay_ms(1000);
     }
     else {
         PORTD = 0; // Turn off 7-segment display
     }
  }  // Endless loop
}
```



Task 5:

Switch Count via 7-Segment Display Write a block of code to count the number of switches that are pressed on PORTC. Display this number on the first digit of the seven segment display, i.e., the digit should reflect a number between zero and eight depending on how many buttons are pressed simultaneously

Lab Report:

Problem Statement:

Write a block of code to count the number of switches that are pressed on PORTC. Display this number on the first digit of the seven-segment display, i.e., the digit should reflect a number between zero and eight depending on how many buttons are pressed simultaneously.

Solution:

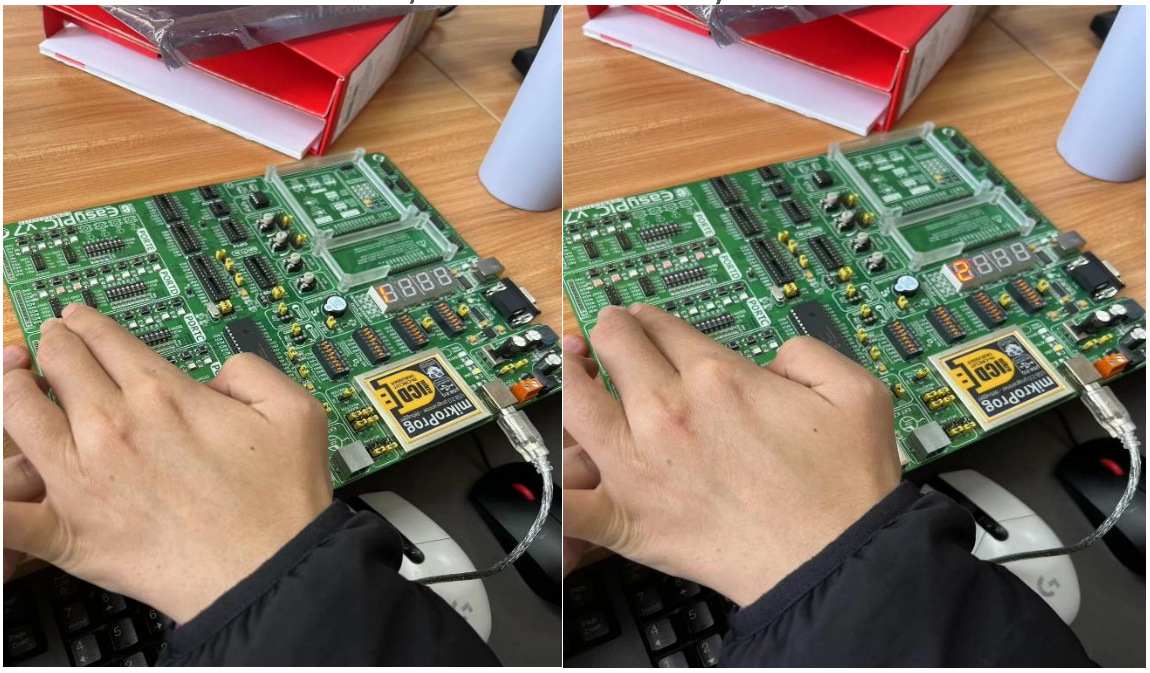
The given code initializes the necessary variables and sets the required configuration for the microcontroller. The code then enters an infinite loop where it counts the number of switches pressed on PORTC and displays the count on the first digit of the seven-segment display. The getDigitCode() function is used to convert the count to the corresponding display pattern.

The count of switches is calculated by iterating over each bit of PORTC and incrementing the switchCount variable if the corresponding bit is set. After calculating the count, the switchCount is cleared, and the pattern for the corresponding digit is retrieved using the getDigitCode() function. The retrieved pattern is then displayed on the first digit of the seven-segment display for a delay of 5ms.

The getDigitCode() function takes a number between 0 and 9 and returns the corresponding pattern to display that number on the seven-segment display. The pattern for each digit is defined using a switch statement.

Code with comments:

```
// Function to retrieve the seven-segment display pattern for a given number
char getDigitCode(int number);

void main() {
    int count = 0;
    unsigned char Pattern = 0, cnt = 0, switchCount = 0;
    // Array to store the seven-segment display patterns for each digit
    unsigned char SEGMENT[] = {0x3f,0x06,0x5B,0x4F,
                               0x66,0x6D,0x7D,0x07,
                               0x7F,0x6F};
    // Configure the required pins as output
    ANSELD = 0;
    TRISD = 0;
    // Configure PORTC as input
    TRISC = 0xFF;

    while(1) {
        // Iterate over each bit of PORTC to count the number of switches
pressed
        count = 0;
        while(count < 8) {
            if(PORTC & (1 << count)) {
                switchCount++;
            }
            count ++;
        }
        // Retrieve the seven-segment display pattern for the count of
switches pressed
        Pattern = getDigitCode(switchCount % 10);
        // Display the pattern on the first digit of the seven-segment
display
        PORTD = Pattern;
        // Delay to display the pattern
        Delay_ms(5);
        // Clear the switch count for the next iteration
        switchCount = 0;
    }
}
char getDigitCode(int number) {
    char digitCode;

    switch(number) {
        case 0:
            digitCode = 0b00111111;   // Display 0
            break;
        case 1:
            digitCode = 0b00000110;   // Display 1
            break;
        case 2:
            digitCode = 0b01011011;   // Display 2
            break;
        case 3:
            digitCode = 0b01001111;   // Display 3
            break;
        case 4:
            digitCode = 0b01100110;   // Display 4
            break;
        case 5:
            digitCode = 0b01101101;   // Display 5
            break;
        case 6:
            digitCode = 0b01111101;   // Display 6
            break;
        case 7:
            digitCode = 0b00000111;   // Display 7
            break;
        case 8:
            digitCode = 0b01111111;   // Display 8
            break;
        case 9:
            digitCode = 0b01101111;   // Display 9
            break;
        default:
            digitCode = 0x00;   // Display blank
            break;
    }

    return digitCode;
}
```



Lab 2

Title: Embedded Systems - Lab Session 2: Digital Clock

Introduction:

In this laboratory session, we aimed to develop our problem-solving skills with embedded systems and further familiarize ourselves with the MikroC development environment, the Proteus simulation environment, the programming languages for PIC, and some basic operations of interrupts and timers/counters. We were tasked with driving a seven-segment display to create a standard 24-hour clock using a PIC18F45K22 microcontroller and the EasyPIC board.

Task 1:

Clock Display

We started by enabling the 4-digit 7-segment display to show hours on the left-most two digits and minutes on the right-most two digits. We added a dot in the middle to indicate the separation between hours and minutes.

Task 2:

Time Setting

We enabled the RB0 button to set the two digits for hours and the two digits for minutes in either the clock setting mode or the alarm setting mode. We programmed the clock setting mode to be enabled or disabled by pressing the RB1 button and the alarm setting mode to be enabled or disabled by pressing the RB2 button.

Task 3:

Clock Setting On/Off

We enabled the RB1 button to enter or quit the clock setting mode. The clock starts at 22:00 when the board is powered on. For the ease of testing, the clock runs at 60 times the normal speed. The value of minutes (or hours) should increase by one step every second (or minute).

Task 4:

Alarm Setting On/Off

We enabled the RB2 button to enter or quit the alarm setting mode. When the RB2 button is pressed, the clock enters the alarm setting mode. Time setting by the RB2 button is similar to that by the RB1 button. When the RB2 button is pressed for the third time to quit the alarm setting mode, the clock runs like an hourglass timer by counting down hours and minutes. When the clock reaches 00:00, an alarm is sounded by activating the piezo buzzer on the board. The clock should get back to normal afterward.

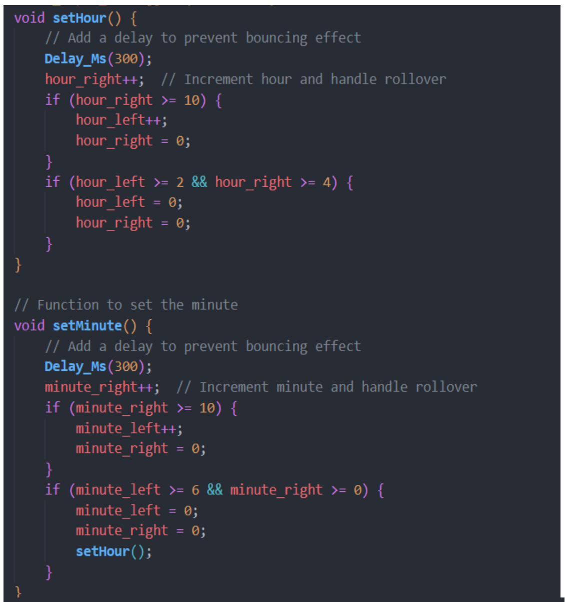
Results:

We successfully completed all four tasks, resulting in a functional digital clock with time setting, clock setting, and alarm setting features. We demonstrated our work during our lab session and answered questions about the project.

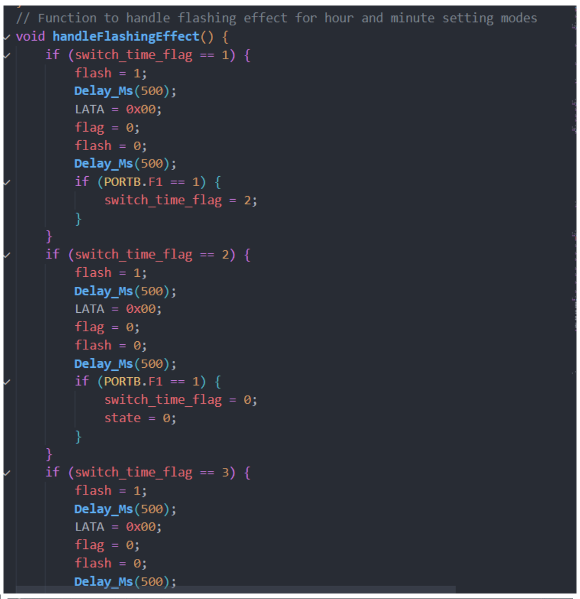
Conclusion:

Through this lab session, we learned how to work with a PIC18F45K22 microcontroller, 7-segment displays, and EasyPIC board to create a digital clock with time and alarm setting features. We gained experience with the MikroC development environment, Proteus simulation environment, and programming languages for PIC.

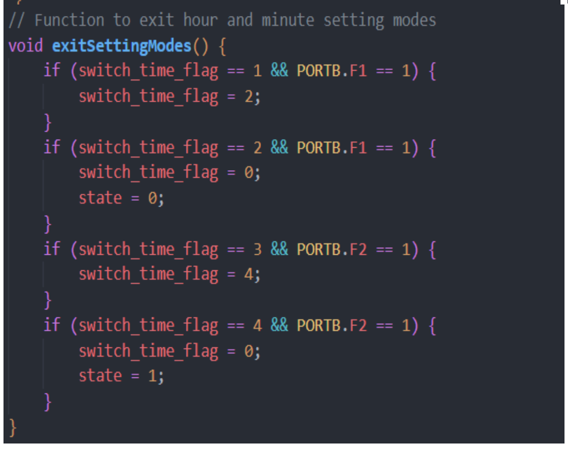
Code:



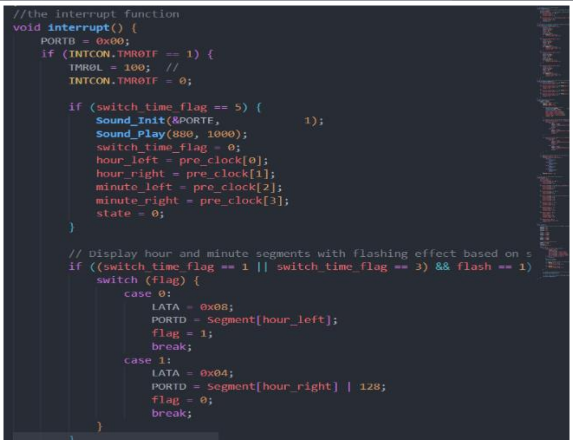
```c
void setHour() {
    // Add a delay to prevent bouncing effect
    Delay_Ms(300);
    hour_right++;  // Increment hour and handle rollover
    if (hour_right >= 10) {
        hour_left++;
        hour_right = 0;
    }
    if (hour_left >= 2 && hour_right >= 4) {
        hour_left = 0;
        hour_right = 0;
    }
}

// Function to set the minute
void setMinute() {
    // Add a delay to prevent bouncing effect
    Delay_Ms(300);
    minute_right++;  // Increment minute and handle rollover
    if (minute_right >= 10) {
        minute_left++;
        minute_right = 0;
    }
    if (minute_left >= 6 && minute_right >= 0) {
        minute_left = 0;
        minute_right = 0;
        setHour();
    }
}
```



```c
// Function to handle flashing effect for hour and minute setting modes
void handleFlashingEffect() {
    if (switch_time_flag == 1) {
        flash = 1;
        Delay_Ms(500);
        LATA = 0x00;
        flag = 0;
        flash = 0;
        Delay_Ms(500);
        if (PORTB.F1 == 1) {
            switch_time_flag = 2;
        }
    }
    if (switch_time_flag == 2) {
        flash = 1;
        Delay_Ms(500);
        LATA = 0x00;
        flag = 0;
        flash = 0;
        Delay_Ms(500);
        if (PORTB.F1 == 1) {
            switch_time_flag = 0;
            state = 0;
        }
    }
    if (switch_time_flag == 3) {
        flash = 1;
        Delay_Ms(500);
        LATA = 0x00;
        flag = 0;
        flash = 0;
        Delay_Ms(500);
```



```c
// Function to exit hour and minute setting modes
void exitSettingModes() {
    if (switch_time_flag == 1 && PORTB.F1 == 1) {
        switch_time_flag = 2;
    }
    if (switch_time_flag == 2 && PORTB.F1 == 1) {
        switch_time_flag = 0;
        state = 0;
    }
    if (switch_time_flag == 3 && PORTB.F2 == 1) {
        switch_time_flag = 4;
    }
    if (switch_time_flag == 4 && PORTB.F2 == 1) {
        switch_time_flag = 0;
        state = 1;
    }
}
```



```c
//the interrupt function
void interrupt() {
    PORTB = 0x00;
    if (INTCON.TMR0IF == 1) {
        TMR0L = 100;  //
        INTCON.TMR0IF = 0;

        if (switch_time_flag == 5) {
            Sound_Init(&PORTE,            1);
            Sound_Play(880, 1000);
            switch_time_flag = 0;
            hour_left = pre_clock[0];
            hour_right = pre_clock[1];
            minute_left = pre_clock[2];
            minute_right = pre_clock[3];
            state = 0;
        }

        // Display hour and minute segments with flashing effect based on s
        if ((switch_time_flag == 1 || switch_time_flag == 3) && flash == 1)
            switch (flag) {
                case 0:
                    LATA = 0x08;
                    PORTD = Segment[hour_left];
                    flag = 1;
                    break;
                case 1:
                    LATA = 0x04;
                    PORTD = Segment[hour_right] | 128;
                    flag = 0;
                    break;
            }
```

This is the code for a digital clock project implemented using a PIC18F45K22 microcontroller and a 4-bit 7-segment display. The following are explanations of the various sections of the code:
Global Variables: A number of global variables such as hours, minutes, flag bits etc. are defined and used to track and manipulate the state of the clock throughout the program.
Segment[] array: This is an array that stores the code corresponding to each digit (0-9) of the 7-segment display.
alarm() function: this is used to decrement the minutes and adjust the hours as required. If both hours and minutes are 0, an alarm will be triggered and the clock will be reset.
setHour() function: sets the hour, increments the hour as required and handles the rollback of the hour.
setMinute() function: sets the minutes, adds minutes as required and handles rollback of minutes.
interrupt() function: this is an interrupt service routine that handles updating and setting the clock. It relies on the global variable ***switch_time_flag*** to determine whether it is currently in display mode or set mode, and performs the appropriate action depending on the mode. In display mode, it updates the 7-segment display with the current hour and minute values. In setup mode, it listens for key inputs to set the hours and minutes.
main() function: This is the entry point of the program. It first configures the I/O ports and then enters an infinite loop. In this loop, the clock switches between display mode and setting mode depending on the key input and the value of ***switch_time_flag.***
The ***exitSettingModes()*** function: is used to switch between hour and minute setting modes, and to exit setting mode.
Overall, this code implements a basic digital clock function, including displaying the current time,

setting the time and triggering an alarm. By dividing the logic into smaller functions, the code becomes easier to read and maintain

Experiment:

Lab3

Task 1:

Writing/Reading Data via UART
Problem description:
The objective of this task is to use the MikroC development environment and the Proteus simulation environment to implement data sending and receiving between a PC and a microcontroller via the UART communication protocol. Specifically, a welcome message needs to be sent from the PC terminal to the microcontroller via the serial port, and then the message needs to be sent back from the microcontroller to the PC terminal for display via the serial port.
Solution:
First, configure the UART module in MikroC, setting the baud rate to 9600, the data bits to 8 and the stop bit to 1. Then define a buffer to store the received data. Then write an initialisation function to enable the UART module. Next, write a function to receive and display a welcome message from the PC terminal and store the message in a buffer for subsequent transmission. Finally, a function is written to send the message in the buffer back to the PC terminal for display.

Task 2

Writing/Reading Data via I2C
Problem description:
The goal of this task is to use the MikroC development environment and the Proteus simulation environment to implement data sending and receiving between the microcontroller and the EEPROM via the I2C communication protocol. Specifically, a 4-digit or alphabetic message needs to be written to the EEPROM, then the message needs to be read from the EEPROM and displayed on a seven-segment digital tube on the EasyPIC board.
Solution:
First, configure the I2C module in MikroC and define a buffer for storing the message. Then write an initialisation function to enable the I2C module. Next, write a function to write the message to the EEPROM and wait a while after the write is complete to ensure that the data has been successfully written to the EEPROM. Finally a function is written to read the message from the EEPROM and convert it to a seven-segment digital tube for display.

Task 3

CLI via UART and I2C
Problem description:
The objective of this task is to implement a Command Line Interface (CLI) using a combination of UART and I2C communication protocols for sending messages entered by the PC terminal to the microcontroller via UART and storing the messages in the EEPROM via I2C. Each byte is then read from the EEPROM and converted into a seven-segment digital tube for display.
Solution:
First, write initialisation functions to enable the UART and I2C modules. Then a CLI function is written to prompt the user for a message and send the message to the UART. Next, write a function to write the message received by the UART into the EEPROM. Finally a function is written to read each byte from the EEPROM and convert it to a seven-segment digital tube for display. Wait for a period of time between each read so that the numbers on the seven-segment digital tube can be displayed clearly.

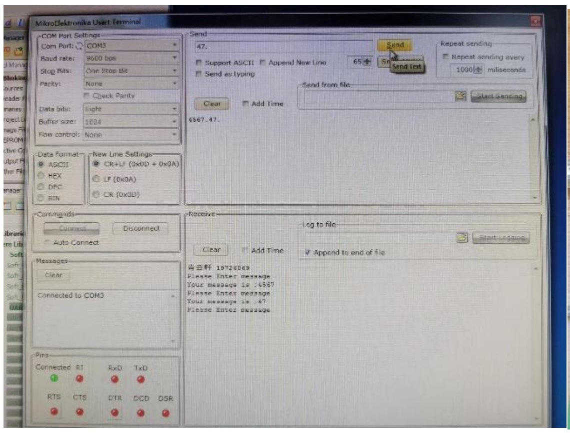
Code

```
// Task1

// Declare global variables
char number[6];
int D1 = 0, D2 = 0, D3 = 0, D4 = 0;
int i = 0, j = 0;
int flag = 0;
unsigned char Segment[] =
{0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};
int uart_rd;
int tagForEnter = 0, tagForDigital = 0;

// Function to update hours
void setHour() {
   // Increment hour units
   D2++;
   if(D2 >= 10){
      // Increment hour tens and reset units
      D1++;
      D2 = 0;
   }
   // Reset both tens and units when reaching 24 hours
   if(D1 >= 2 && D2 >= 4){
      D1 = 0;
      D2 = 0;
   }
}

// Function to update minutes
void setMinute() {
   // Increment minute units
   D4++;
   if(D4 >= 10){
      // Increment minute tens and reset units
      D3++;
      D4 = 0;
   }
   // Reset both tens and units when reaching 60 minutes
   if(D3 >= 6 && D4 >= 0){
      D3 = 0;
      D4 = 0;
      setHour();  // Update hours when minutes reset
   }
}

// Interrupt handler function
void interrupt() {
  if(INTCON.TMR0IF == 1){
     TMR0L = 100; //
     INTCON.TMR0IF = 0;
     if(2000<= j){
        setMinute();
        j = 0;
     }
     else j++;

     // Update the 7-segment display based on the current flag value
     switch(flag) {
        case 0:
            LATA = 0x08;
            PORTD = Segment[D1];
            flag = 1;
            break;
        case 1:
            LATA = 0x04;
            PORTD = Segment[D2];
            flag = 2;
            break;
        case 2:
            LATA = 0x02;
            PORTD = Segment[D3];
            flag = 3;
            break;
        case 3:
            LATA = 0x01;
            PORTD = Segment[D4];
            flag = 0;
            break;
      }
   }
}

// Main function
void main() {
   // Configure all pins as digital I/O
   ANSELA = ANSELB = ANSELC = ANSELD = ANSELE = 0;

   // Configure PORTB, PORTA, and PORTD as outputs
   TRISB = TRISA = TRISD = 0x00;

   PORTB = 0x06;
   T0CON = 0xC4;
   GIE_bit = 1;
   TMR0IE_bit = 1;
   RCSTA1.CREN = 1;

   // Initialize UART module at 9600 bps
   UART1_Init(9600);
   Delay_ms(100);  // Wait for UART module to stabilize

   // Send "Hello World!" message via UART
   UART1_Write_Text("Hello World!");
   UART1_Write(13);
   UART1_Write(10);
    // Main loop
    while(1) {
    // Check if data is received through UART
    if (UART1_Data_Ready()) {
    tagForEnter = 0;
    uart_rd = UART1_Read(); // Read the received data
    number[i] = uart_rd;
    i++;
```

```
1_Write(10);
// Main loop
while(1) {
// Check if data is received through UART
if (UART1_Data_Ready()) {
tagForEnter = 0;
uart_rd = UART1_Read(); // Read the received data
number[i] = uart_rd;
i++;
      // Initialize I2C communication
  I2C1_Init(100000);
  I2C1_Start();
  I2C1_Wr(0xA2);             // Send byte via I2C (device address + W)
  I2C1_Wr(2);                // Send byte (address of EEPROM location)
  I2C1_Wr(uart_rd);          // Send data (data to be written)
  I2C1_Stop();               // Issue I2C stop signal

  I2C1_Start();              // Issue I2C start signal
  I2C1_Wr(0xA2);             // Send byte via I2C (device address + W)
  I2C1_Wr(2);                // Send byte (data address)
  I2C1_Repeated_Start();     // Issue I2C signal repeated start
  I2C1_Wr(uart_rd);          // Send byte (device address + R)
  I2C1_Stop();               // Issue I2C stop signal

  // Convert the received character to the corresponding digit or letter
  if(48 <= uart_rd && uart_rd <= 57) uart_rd  = uart_rd - '0';
  if(65 <= uart_rd && uart_rd <= 90)  uart_rd  = uart_rd - 'A' + 10;
  if(97 <= uart_rd && uart_rd <= 122) uart_rd  = uart_rd - 'a' + 10;

  // Update the 7-segment display based on the received data
  switch(tagForDigital) {
      case 0:
          D1 = 0, D2 = 0, D3 = 0, D4 = 0;
          D1 = uart_rd;
          tagForDigital = 1;
          break;
      case 1:
          if(uart_rd == 46) {
              tagForDigital = 0;
              tagForEnter = 0;
              i = 0;
              D4 = D1;
              D1 = 0;
              UART1_Write_Text("Your message is :");
              UART1_Write(number[0]);
              UART1_Write(13);
              UART1_Write(10);
              break;
          }
          D2 = uart_rd;
          tagForDigital = 2;
          break;
      case 2:
          if(uart_rd == 46) {
              tagForDigital = 0;
              tagForEnter = 0;
              i = 0;
              D3 = D1;
              D4 = D2;
              D1 = 0;
              D2 = 0;
              UART1_Write_Text("Your message is :");
              UART1_Write(number[0]);
              UART1_Write(number[1]);
              UART1_Write(13);
              UART1_Write(10);
              break;
          }
          D3 = uart_rd;
          tagForDigital = 3;
          break;
      case 3:
          if(uart_rd == 46) {
                D4 = D3;
                D3 = D2;
                D2 = D1;
                D1 = 0;
                tagForDigital = 0;
                tagForEnter = 0;
                i = 0;
                UART1_Write_Text("Your messageis :");
                UART1_Write(number[0]);
                UART1_Write(number[1]);
                UART1_Write(number[2]);
                UART1_Write(13);
                UART1_Write(10);
                break;
            }
            D4 = uart_rd;
            tagForDigital = 4;
            break;
     case 4:
            i = 0;
            tagForEnter = 0;
            tagForDigital = 0;
            UART1_Write_Text("Your message is :");
            UART1_Write(number[0]);
            UART1_Write(number[1]);
            UART1_Write(number[2]);
            UART1_Write(number[3]);
            UART1_Write(13);
            UART1_Write(10);
```

```
                    break;
                }
            }
        }
}
```

Experiment



```
Com Port: COM3
Baud rate: 9600 bps
Stop Bits: One Stop Bit
Parity: None
Data bits: Eight
Buffer size: 1024
Flow control: None
Data Format: ASCII
New Line Settings: CR+LF (0x0D + 0x0A)
Send: 47.
Repeat sending every 1000 miliseconds
4567.47.
Connected to COM3
Append to end of file

肖云轩 19726069
Please Enter message
Your message is :4567
Please Enter message
Your message is :47
Please Enter message
```