MICROPROCESSOR BASED SYSTEM DESIGN

Complex Engineering Problem

CS-20093 Ammar Lodhi
CS-20096 Ubaidullah Amir
CS-20103 Umer Ahmed

DEPARTMENT OF COMPUTER & INFORMATION SYSTEMS ENGINEERING
BACHELORS IN COMPUTER SYSTEMS ENGINEERING
Course Code: CS-301
Course Title: Microprocessor Based System Design
Complex Engineering Problem
TE Batch 2020, Spring Semester 2023
Grading Rubric
TERM PROJECT Group

Members:

CS-20093	Ammar lodhi
CS-20096	Ubaidullah Amir
CS-20103	Umer Ahmed

CRITERIA AND SCALES

Criterion 1: Does the microcontroller application meet the desired specifications and produce the desired outputs? [CPA 1, CPA 2]

1	2	3	4
The application does not meet the desired specifications and is producing incorrect outputs.	The application partially meets the desired specifications and is producing incorrect or partially correct outputs.	The application meets the desired specifications but is producing incorrect or partially correct outputs.	The application meets all the desired specifications and is producing correct outputs.

Criterion 2: What is the student's level of confidence with handling of equipment? [CPA 3]

1	2	3	4
The student is unfamiliar with the equipment and tools.	The student is aware of equipment and tools to some extent but can't use them adequately	The student is aware of tools and equipment but only partially confident in handling them	The student is proficient with the equipment usage.

Criterion 3: How well the student managed various issues during solution design? [CPA 1, CPA 2, CPA 3]

1	2	3	4
No conflicting issues were managed or resolved at all.	Student could handle only some of the issues but not in a befitting manner.	Most of the issues were handled gracefully	All such issues were resolved with successful implementation.

Criterion 4: Has the student submitted all the deliverables on time? [CPA 1, CPA 2]

1	2	3	4
Only one deliverable submitted	Only few deliverables are submitted.	All deliverables are submitted but not up to the mark and not on time.	All of the deliverables were submitted timely

Total Marks:

Teacher's Signature: _____

Contents

Abstract

The project aims to implement a microcontroller-based system to control the speed of a fan using Pulse Width Modulation (PWM) technique, which varies the duty cycle based on the input temperature provided through a keypad. The fan speed is categorized into three ranges: fast, medium, and slow, determined by predefined temperature thresholds. The current temperature and fan speed are displayed on a seven-segment display. Additionally, a dip switch mechanism is incorporated to manually switch the fan on or off, with the status displayed using LEDs. The design ensures robustness by handling incorrect or invalid temperature ranges.

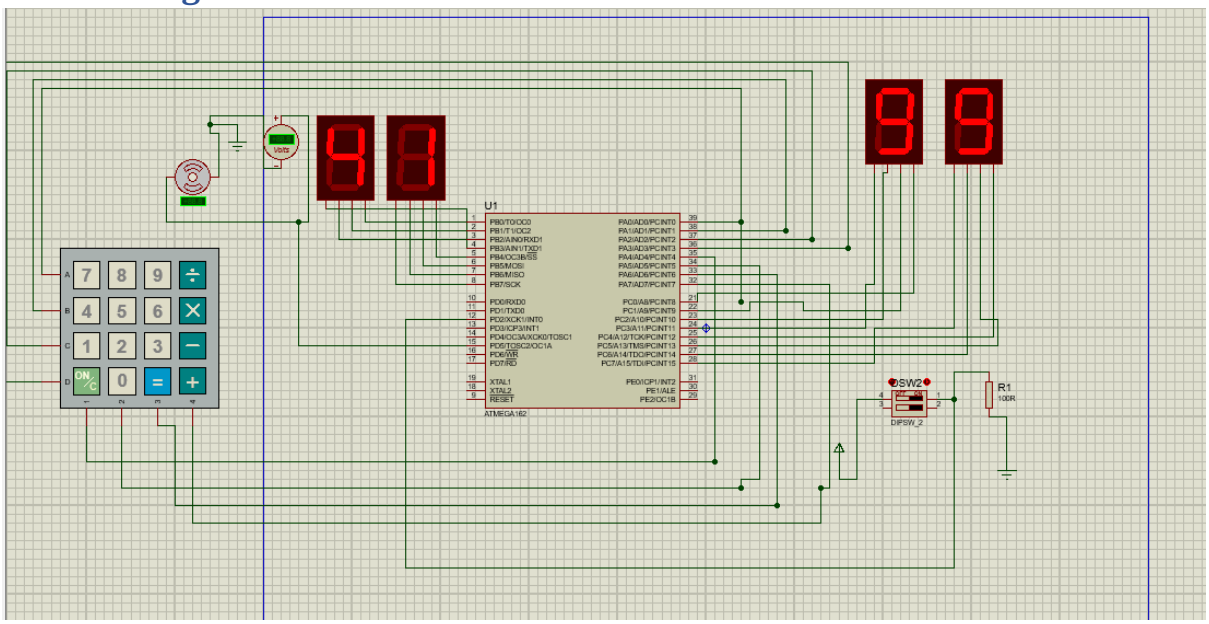
Introduction

The purpose of this project is to develop an intelligent fan control system that can efficiently adjust the fan speed according to the surrounding temperature. By doing so, we can achieve improved energy efficiency and maintain a comfortable environment in various applications, such as rooms, offices, and industrial settings.

Components Used

- Microcontroller
- Keypad
- Seven-Segment Display
- Fan
- LEDs
- Dip Switch
- Power Supply

Circuit Diagram



Working Principle

The microcontroller continuously reads the input temperature from the temperature sensor and processes it. The user can input desired temperature thresholds for the fast, medium, and slow fan speed ranges using the keypad. The microcontroller then determines the appropriate fan speed based on these thresholds using PWM.

The fan speed control algorithm compares the current temperature with the defined thresholds. If the temperature falls within a specific range, the corresponding fan speed is set using PWM duty cycles. The fan operates at full speed in the fast range, a reduced speed in the medium range, and the lowest speed in the slow range.

The seven-segment display shows the current temperature and the corresponding fan speed. Additionally, the dip switch allows manual control of the fan's power state, and LEDs indicate whether the fan is turned on or off.

Software Implementation

The microcontroller is programmed using C programming language. The program utilizes libraries for interfacing with the keypad, temperature sensor, display, and PWM output control. It continuously monitors the input temperature, reads the keypad inputs, and calculates the appropriate PWM duty cycle to set the fan speed.

To handle incorrect or invalid temperature ranges, the program includes error-checking routines and safe default settings to prevent erratic behavior or system failure.

Code

```
#include <avr/io.h>
#include <util/delay.h>

#define PWM_PIN     PD5   // PWM output pin (OC1A)
#define PWM_FREQ    100   // PWM frequency in Hz

#define F_CPU 8000000UL   // Assuming an 8 MHz clock frequency

#define KEYPAD_PORT PORTA
#define KEYPAD_DDR  DDRA
#define KEYPAD_PIN  PINA

#define BUTTON_PIN  PD2   // ON/OFF button pin

// Keypad layout
const unsigned char KEYS[4][4] = {
    {'7', '8', '9', 'A'},
    {'4', '5', '6', 'B'},
    {'1', '2', '3', 'C'},
    {'*', '0', '#', 'D'}
};

void PWM_Init() {
    // Set PD5 (OC1A) as output pin
    DDRD |= (1 << PWM_PIN);

    // Set fast PWM mode with non-inverted output
    TCCR1A = (1 << COM1A1) | (1 << WGM11);
    TCCR1B = (1 << WGM13) | (1 << WGM12) | (1 << CS10);
    // Set PWM frequency
    ICR1 = F_CPU / (2 * PWM_FREQ);
}

void PWM_SetDutyCycle(uint8_t dutyCycle) {
    // Calculate the OCR1A value based on the duty cycle
    uint16_t ocrValue = (dutyCycle / 100.0) * ICR1;

    // Set the new duty cycle value
    OCR1A = ocrValue;
```

```
}

void runMotorSpeed(int temp) {
    if (temp < 25) {
        // Run the motor slowly
        // Set PC0-PC3 to output the value 2 (binary: 0010)
        PORTC = (PORTC & 0xF0) | 0x02;
        // Set PC4-PC7 to output 0 (binary: 0000)
        PORTC &= 0x0F;
        PWM_SetDutyCycle(20);    // 10% duty cycle
        } else if (temp >= 25 && temp <= 40) {
        // Increase motor speed
        // Set PC0-PC3 to output the value 5 (binary: 0101)
        PORTC = (PORTC & 0xF0) | 0x05;
        // Set PC4-PC7 to output 0 (binary: 0000)
        PORTC &= 0x0F;

        PWM_SetDutyCycle(50);    // 50% duty cycle
        } else {
        // Run the motor full speed
        // Set PC0-PC3 to output the value 9 (binary: 1001)
        PORTC = (PORTC & 0xF0) | 0x09;

        // Set PC4-PC7 to output the value 9 (binary: 1001)
        PORTC = (PORTC & 0x0F) | 0x90;
        PWM_SetDutyCycle(100);    // 100% duty cycle
    }
}

// Function to initialize the keypad
void keypad_init() {
    // Set the keypad port as input
    KEYPAD_DDR = 0x0F;

    // Enable pull-up resistors
    KEYPAD_PORT = 0xFF;
}

// Function to read the keypad
char keypad_read() {
    // Loop through each column
    for (uint8_t col = 0; col < 4; col++) {
        // Set current column as output low
        KEYPAD_DDR = (0b00010000 << col);
        KEYPAD_PORT &= ~(0b00010000 << col);  // Set current column as input
with pull-up

        // Delay for a short period for stabilization
```

```
        _delay_ms(1);

        // Loop through each row
        for (uint8_t row = 0; row < 4; row++) {
            // Check if the key in the current row and column is pressed
            if (!(KEYPAD_PIN & (0b00000001 << row))) {
                // Return the corresponding character from the KEYS array
                return KEYS[row][col];
            }
        }

        // Set current column as input high (disable pull-up)
        KEYPAD_PORT |= (0b00010000 << col);
    }

    // Return null character if no key is pressed
    return '\0';
}

// Function to check if a character is a digit
int is_digit(char c) {
    return (c >= '0' && c <= '9');
}

// Function to convert a digit character to its integer value
int char_to_int(char c) {
    return c - '0';
}
//disables the motor and the displays
void cleanUp(){
    // Disable everything and start from the beginning
    _delay_ms(1000);   // Delay for button debounce
    PORTB = 0x00;
    PWM_SetDutyCycle(0);  // Reset motor speed
    PORTC=0x00;
}
// Function to read the keypad and return a single-digit input
char get_single_digit() {
    while (1) {
        // Check the state of the ON/OFF button
        if ((PIND & (1 << BUTTON_PIN)) == 0) {
            // Button is pressed (OFF condition)
            cleanUp();  // Perform clean up and restart the code
        }

        // Read the keypad
        char key = keypad_read();
```

```
        // Check if a key is pressed and if it is a digit
        if (key != '\0' && is_digit(key)) {
            // Return the single-digit key input
            return key;
        }
    }
}

int main(void) {
    // Initialize the keypad
    keypad_init();
    // Set PORTB as key inputs
    DDRB = 0xFF;
    //set PORTC as output for duty cycle
    DDRC = 0xFF;
    // Initialize PWM
    PWM_Init();

    while (1) {
        // Get the first single-digit input
        char firstDigit = get_single_digit();


        // Wait for the second single-digit input
        char secondDigit;
        // Display the lower 4 bits of the input1 on PORTA
        PORTB = (PORTB & 0xF0) | (char_to_int(firstDigit) & 0x0F);

        _delay_ms(500);
        while (1) {
            // Check the state of the ON/OFF button
            if ((PIND & (1 << BUTTON_PIN)) == 0) {
                // Button is pressed (OFF condition)
                cleanUp();  // Perform clean up and restart the code
            }

            secondDigit = get_single_digit();
            if (secondDigit != '\0') {
                break;
            }
        }

        // Display the higher 4 bits of the variable on PORTA
        PORTB = (PORTB & 0x0F) | ((char_to_int(secondDigit) & 0x0F) << 4);

        _delay_ms(500);

        // Create the double-digit number
```

```
        int doubleDigit = char_to_int(firstDigit) * 10 +
char_to_int(secondDigit);


        // Run the motor
        runMotorSpeed(doubleDigit);
    }

    return 0;
}
```

Here is the brief explanation of each function.

1. PWM_Init():
The PWM_Init() function is responsible for initializing the Pulse Width Modulation (PWM) module on the AVR microcontroller. It sets the necessary configuration for PWM operation, such as setting the PWM output pin (PD5), selecting Fast PWM mode with non-inverted output on OC1A, and configuring the prescaler to generate the desired PWM frequency. The PWM frequency is set based on the F_CPU and PWM_FREQ constants. This function prepares the microcontroller to generate variable duty cycle PWM signals that will be used to control the motor speed.

2. PWM_SetDutyCycle(uint8_t dutyCycle):
The PWM_SetDutyCycle() function is used to set the duty cycle of the PWM signal. It takes a single parameter 'dutyCycle' representing the desired percentage duty cycle, ranging from 0 to 100. The function calculates the corresponding OCR1A value based on the duty cycle percentage and adjusts the duty cycle accordingly. This allows the motor speed to be adjusted dynamically by varying the duty cycle of the PWM signal generated by the PWM module.

3. runMotorSpeed(int temp):
The runMotorSpeed() function is responsible for controlling the motor speed based on the temperature input 'temp'. The function takes an integer value 'temp' representing the current temperature. Based on the temperature value, the function selects one of three speed levels for the motor: slow, medium, or fast. It sets the appropriate configuration on PORTC, which is connected to the motor driver inputs, to control the motor direction and speed. The function also calls the PWM_SetDutyCycle() function to adjust the PWM duty cycle, effectively controlling the motor speed based on the temperature range.

4. keypad_init():
The keypad_init() function initializes the 4x4 matrix keypad. It sets KEYPAD_DDR to output on the low nibble (bits 0-3) and sets KEYPAD_PORT with pull-up resistors on the high nibble (bits 4-7) to read the keypad inputs. This function prepares the microcontroller to read the pressed keys from the keypad and detect user input.

5. keypad_read():
The keypad_read() function is used to read a key press from the 4x4 matrix keypad. It loops through each column and row of the keypad, setting the current column as output low and checking for key presses in each row. If a key is pressed, the function returns the corresponding character from the KEYS array, representing the value of the pressed key. If no key is pressed, the function returns the null character ('\0').

6. is_digit(char c):

The is_digit() function checks if a given character 'c' is a digit (0-9). It returns an integer value (0 or 1) indicating whether the character is a digit or not. This function is used to validate the keypad input and ensure that only single-digit keys are accepted for motor speed adjustment.

7. char_to_int(char c):
The char_to_int() function converts a digit character 'c' to its corresponding integer value. It subtracts the ASCII value of '0' from the character 'c' to obtain the integer representation of the digit. The function returns the integer value of the digit, which is used in various parts of the code to process and manipulate the user input.

8. cleanUp():
The cleanUp() function is called when the ON/OFF button is pressed to disable the motor and reset the displays. It first introduces a delay for button debounce, then sets the motor speed to 0 by calling PWM_SetDutyCycle(0). Additionally, it resets the display values on PORTB and clears the display values on PORTC. This function prepares the system for a fresh start when the motor is turned off.

9. get_single_digit():
The get_single_digit() function reads a single-digit input from the keypad. It enters an infinite loop and waits for the user to press a single-digit key on the keypad. While waiting for the input, the function constantly checks the state of the ON/OFF button. If the button is pressed (OFF condition), the cleanUp() function is called to disable the motor and reset the displays. Once a single-digit key is pressed on the keypad, the function returns the corresponding character representing the input digit.

The main() function utilizes these functions to control the motor speed based on user input from the keypad, while also displaying the input and motor speed on the corresponding ports. The motor speed is adjusted using PWM signals generated through PWM_Init() and PWM_SetDutyCycle() functions, while the keypad input is processed using the keypad_read() and get_single_digit() functions. The ON/OFF button is checked periodically to handle the motor's on/off state and reset the system accordingly.

Results

The project successfully implements the microcontroller-based fan speed control system. It accurately reads the input temperature, displays it on the seven-segment display, and adjusts the fan speed accordingly.

The fan speed changes smoothly as the temperature transitions between the predefined ranges. The dip switch allows the user to manually control the fan's power state, and the LEDs provide clear status indications.

Conclusion

The project demonstrates a practical and efficient solution for fan speed control based on temperature sensing. The microcontroller efficiently processes temperature data and performs PWM to regulate the fan's speed in real-time. The added dip switch and LEDs enhance user control and provide status feedback.

This intelligent fan control system has potential applications in various scenarios where maintaining a comfortable environment with energy efficiency is vital.

Future Enhancements

- Implement a feedback loop using an external temperature sensor for improved accuracy.
- Integrate wireless communication to enable remote control and monitoring of the fan system.
- Explore advanced fan speed control algorithms for smoother transitions between speed ranges.
- Incorporate safety features, such as fan stall detection or temperature limit protection.