**Acknowledgments**

The success and final outcomes of this project required a lot of guidance and assistance from many people and the authors are extremely fortunate to have got this all along the completion of our project work. The authors also thank and appreciate the co-workers in developing this project and people who willingly helped as out with their abilities and owe profound gratitude to all the professors of Department of Computer Information and Systems Engineering for their proper guidance and motivation for this project. Last but not the least, the authors are extremely thankful to mam **Syeda Ramish Fatima,** for always being there for us and guiding us throughout the semester.

**PROBLEM STATEMENT**

DESIGN AND IMPLEMENT A MICROCONTROLLER/MICROPROCESSOR-BASED

APPLICATION FOR A REAL-WORLD SCENARIO.

**Task Description**

Implement a microcontroller application for a temperature-controlled DC Fan Motor using various resources. The fan motor must be operated at different speed limits according to some threshold values of ambient room temperature. The temperature sensor output is analog signal hence an ADC is to be interfaced with the microcontroller. Speed control must be achieved using Pulse Width Modulation (PWM) technique.

Additionally, the application must have a provision of manually switching the fan ON and OFF, bypassing the temperature sensor results.

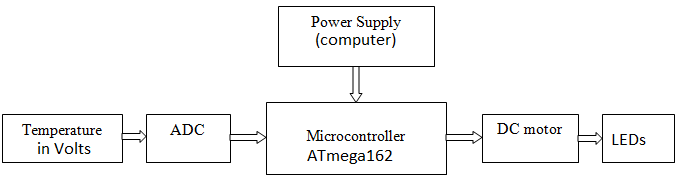
**Summary Of Project**

In this project, we’ve created a temperature-controlled DC fan which includes;

* Codingof our project using C#
* Software simulation of working project
* Hardware of our project

Here, we are taking input temperature as voltages using potentiometer and sending it to ADC0808. Analog signals are then converted into digital where we used Pulse Width Modulation (PWM) logic and sent the data to our micro-controller. At output we have connected a DC fan that is receiving input from the micro-controller. In this way, we control the speed of our DC fan using variable inputs through potentiometer.

**Schematic Block Diagram**



**Introduction**

A Temperature Controlled DC Fan is a system which automatically turns on a DC Fan when the ambient temperature increases above a certain limit. The proposed system temperature-controlled fan using microcontroller is used to control the speed of the fan according to the temperature and specify the temperature in the display.

**How Temperature Controlled DC Fan Circuit using Microcontroller Works?**

1. Initially switch the power supply.
2. Microcontroller starts reading the voltage inputs through potentiometer.
3. This analog value is applied to the analog to digital converter pin of the ADC0808.
4. When the temperature is greater than the threshold value that is to be controlled by a switch, microcontroller sends a command to the controller to switch the motor/fan.
5. Thus, fan starts rotating.

**Resources**

The required components are stated below.

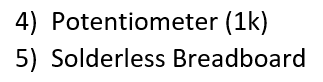
# **Software Section:**

1. Atmel Studio 7.0 (source code in programming language C#)
2. Proteus 8 professional (simulation)

# **Micro-controller Simulation Section:**

1. ATmega162 microcontroller
2. POT-Hg
3. 1ROW-4COL-RED
4. 3WATT100R
5. FAN-DC/DC-MOTOR
6. 10WATT1K
7. DIPSW\_2

# **Hardware Section:**

1. Micro-controller ATmega162
2. DC-FAN/DC-MOTOR
3. ADC0808

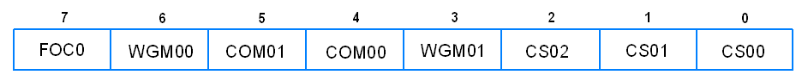
# **Basic Principle**

The project works on the principle of Analog to Digital Conversion. The Analog data from the potentiometer (that works as a temperature sensor) is given to the analog to digital converter ADC0808 and that is an 8-bit ADC.

As per the changes in the temperature, the output of the ADC is generated. The digital output of the ADC is given to Microcontroller to calculate the temperature and control the fan accordingly.

**PWM generation by Phase correct PWM mode:**

The Phase correct PWM mode can be selected by assigning bits WGM0[1:0]=01. This mode is based on dual slope operation. In dual slope operation, TCNTn counts from bottom value to maximum value and maximum value to bottom value. The Output Compare Register compares the value with the TCNTn register constantly during up-counting and down-counting. On compare match PWM output pin (OCn) behaves according to our given logic i.e. as the OCR compares the value of COM0 it constantly becomes low.



We’ve set our Compare Output Match action according to following table:





*Clock Source Select* is set as stated below:

These bits are used to select a clock source. When CS02: CS00 = 000, then timer is stopped. As it gets value between 001 to 101, it gets clock source and starts as the timer. 

Here, we have selected the mode 0 1 0 in which the clock value is divided by 8 that gives us the number of ticks per clock in our project.

# **Working**

The aim of this project is to design a temperature-controlled fan using AVR ATmega16 micro-controller, in which the fan is turned ON or OFF and according to the temperature(voltage) it changes its speed.

* In this circuit, the POT-HG potentiometer is used in place of a temperature sensor that will give continuous analog output corresponding to the voltage sensed by it.
* This analog signal is given to the ADC0808, which converts the analog values to digital values.
* The digital output of the ADC0808 is equivalent to sensed analog voltage.
* In order to get the temperature from the sensed analog voltage, we need to perform some calculations in the programming for the microcontroller. Once the calculations are done by the microcontroller according to the logic, using Pulse Width Modulation the microcontroller controls the DC fan maintaining the variable speed.
* Before the microcontroller gets the PWM data, we’ve used dipswitches to maintain 3 modes:

1. 0 0 Clock is disabled having duty cycle zero . Thus, fan is continuously off.
2. 0 1 Duty cycle is of 50%.
3. 1 0 PWM user defined values work having variable duty cycles such as 10-30 %, 30-60%, and 60% onwards.

Here, we have assumed that 0% is 0 volts and 100% is 100 volts given by the potentiometer.

**The Code Section:**

#include <avr/io.h>

#define *F\_CPU* 1000000UL

#include <util/delay.h>

#include "avr/iom162.h"

void adc(void);

void read\_adc(void);

int main(void)

{

DDRC = 0x00; //input of micro-controller from ADC

DDRE = 0xFF; //output to ADC select interrupt

DDRB = 0x77; //PB0 = OC0 PB1 = OC2 PB2 = (unconnected) PB3 = EOC PD4 = OE PB5 = ALE PB6 = START PB1 = OC2

DDRD = 0xFF; // output to LED

DDRA = 0x00;

TCCR0|= (1<<WGM01)|(1<<COM00)|(1<<CS00); // WGM = 10 CTC

// COM= 01 toggle on compare clear

//CS = 001 no pre-scaler.

TCCR2|= (1<<CS21)|(1<<WGM20)|(1<<COM21); //CS = 010 8 pre-scaler WGM = 10 phase correct PWM

//|(1<<COM20)|(1<<CS22); //|(1<<WGM21)

// COM= 10 clear OCR2 on compare match (inverted)

OCR0|= 2; // 2 cycles 1 pulse

PORTB|= 0x08; //OE set to 1

TIMSK = 0x00; //Timer Interrupt mask

while (1)

{

adc();

}

}

void adc(void)

{

PORTE = 0x00; //IN0 set

read\_adc();

}

void read\_adc()

{

char SW;

unsigned char num;

PORTB = PORTB| 0x60; // ALE and START set to 1

*\_delay\_ms*(100);

PORTB = PORTB & 0x2F; // ALE and START reset to 0

while((PINB&0x08)==0x08);

while((PINB&0x08)==0x00);

PORTB = PORTB | (0x10); // OE set to 1

*\_delay\_ms*(200);

num = PINC;

PORTD = num; //converted bits send to PORTD

PORTB = PORTB&(0xEF); // OE reset to 0

*\_delay\_ms*(100);

SW = PINA & 0x03;

switch(SW)

{

case 0: //fan off

{

OCR2 = 0;

break;

}

case 1: //fan on

{

OCR2 = 128; //50% duty cycle

break;

}

case 2: //Adjusted Values

{

if (num==0x00)

{

OCR2 = 0; //0% duty cycle

}

else if (num>=0x02 && num<=0x4F)

{

OCR2 = 62; //24% duty cycle

}

else if (num>=0x52 && num<=0x9F)

{

OCR2 = 102; //40% duty cycle

}

else if (num>=0xA1 && num<=0xEF)

{

OCR2 = 192; // 75% duty cycle

}

else if (num>=0xF0 && num>=0xFF)

{

OCR2 = 230; //90% duty cycle

}

break;

}

default:

{

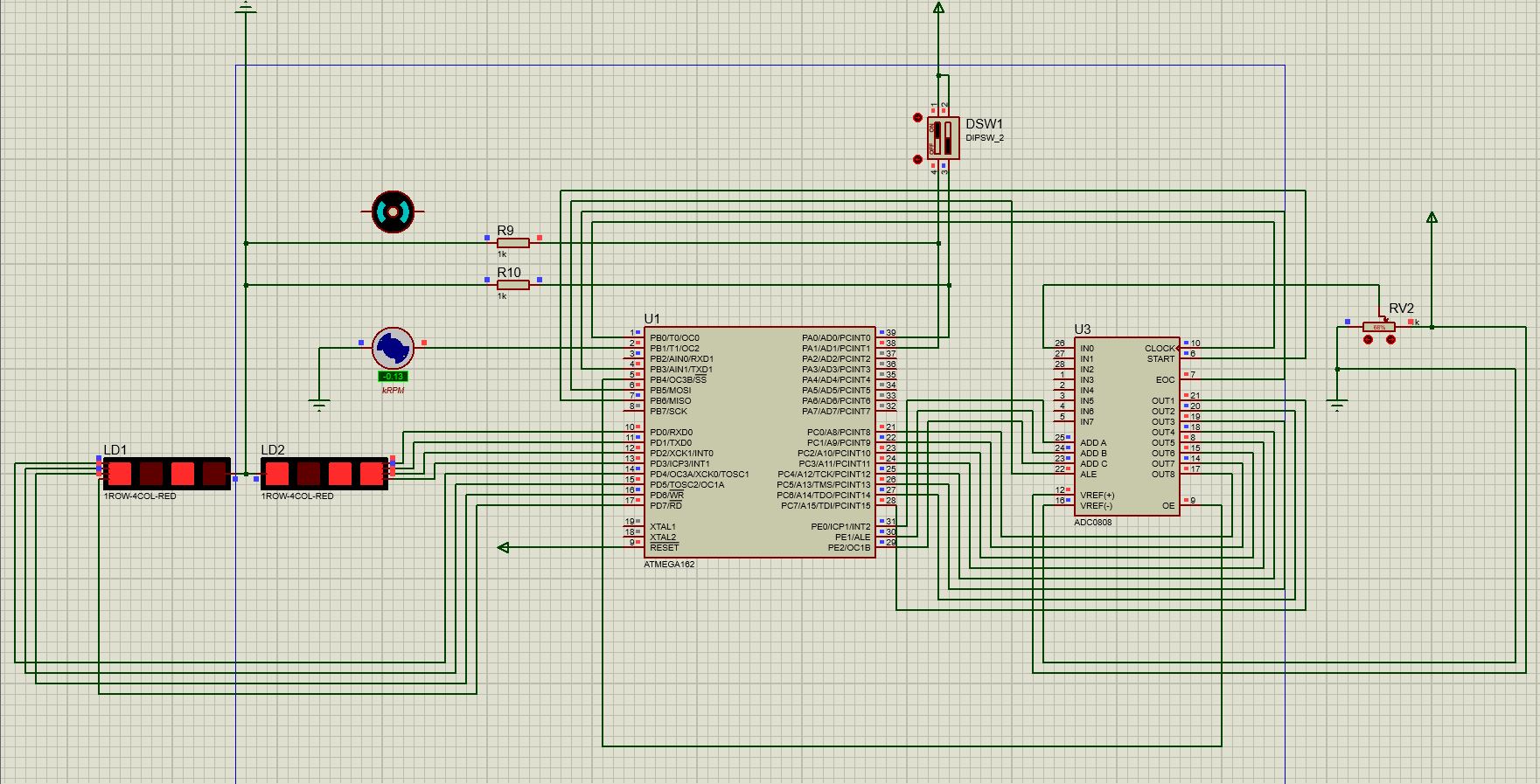
OCR2 = 0;

break;

}

}

}

**Working Simulation:**

Here, we have created software simulation on Proteus 8 professional. In which, we’ve used the components as stated above already.

**Adjusting the Duty Cycles**



For example: if duty cycle is 75%

OCR value = (75÷100) × 256

Resolution: Minimum time of a timer that a counter can measure can be calculated by

Range can be calculated by

where n= size of counter/timer

Pre-Scaling is calculated by

Where pre-scaler is used to increase the range.

**Defining PORTS Of AVR ATmega16 Micro-controller**

**PORTA:**

It is defined for the user interface. To get which mode is to be selectedON, OFF, or variable speed (using if and else conditions).

**PORTB:**

It’s used for the ADC conversion. PORTB is also used to give the output of TIMER-COUNTER.

**PORTC:**

It acts as input port. It is used to take the converted output from the ADC.

**PORTD:**

It is connected to LEDs. So that the output combinations are also visible through the LEDs.  
**PORTE:**

It is used to select the different sensors or hardware we’ve used such as DC-FAN/DC-MOTOR or potentiometer.

**Applications**

* Temperature Controlled DC Fan can be used to control the temperature of devices, rooms, electronic components etc. by monitoring the temperature.
* Can be extended to PWM based output, where the speed of the fan can be varied according to the duty cycle of the PWM signal.
* The circuit can be used in CPU to reduce the heat.

**Conclusion**

We interfaced the ADC with AVR ATmega16 microcontroller as instructed by ma’m Ramish Fatima taking help from lab exercises such as lab6, lab9, etc. The most challenging task was to use PWM into our project to maintain the speed of the fan.