Mini Project 3

System Requirements

Implement the following **Fan Controller** system with the specifications listed below:

1. The aim of this project is to design a temperature-controlled fan using **ATmega32** microcontroller, in which the fan is automatically turned ON or OFF according to the temperature. Use **ATmega32** Microcontroller with frequency **1Mhz**.

2. In this project, the LM35 temperature sensor will give continuous analog output corresponding to the temperature sensed by it. This analog signal is given to the ADC, which converts the analog values to digital values.

3. The digital output of the ADC is equivalent to sensed analog voltage.

4. In order to get the temperature from the sensed analog voltage, we need to perform some calculations in the programming for the microcontroller.

5. Once the calculations are done by the microcontroller according to the logic, the temperature is displayed on the LCD.

6. The microcontroller will continuously monitor the temperature and based on the temperature value, the microcontroller will drive the fan like that:

    a. If the temperature is less than 30C turn off the fan.

    b. If the temperature is greater than or equal 30C turn on the fan with 25% of its maximum speed.

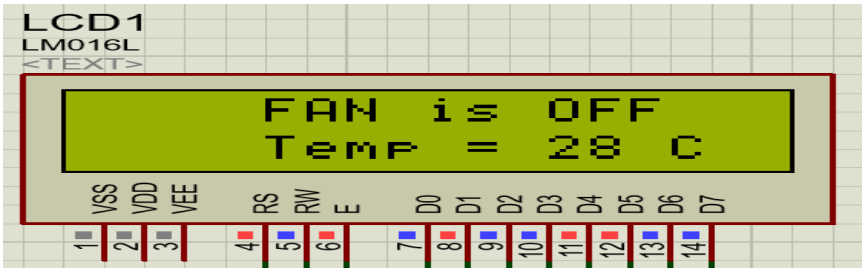
    c. If the temperature is greater than or equal 60C turn on the fan with 50% of its maximum speed.

    d. If the temperature is greater than or equal 90C turn on the fan with 75% of its maximum speed.

e. If the temperature is greater than or equal 120C turn on the fan with 100% of its maximum speed.

7. The main principle of the circuit is to switch on/off the fan connected to DC motor based on temperature value. **The DC-Motor rotates in clock-wise direction or stopped based on the fan state.**

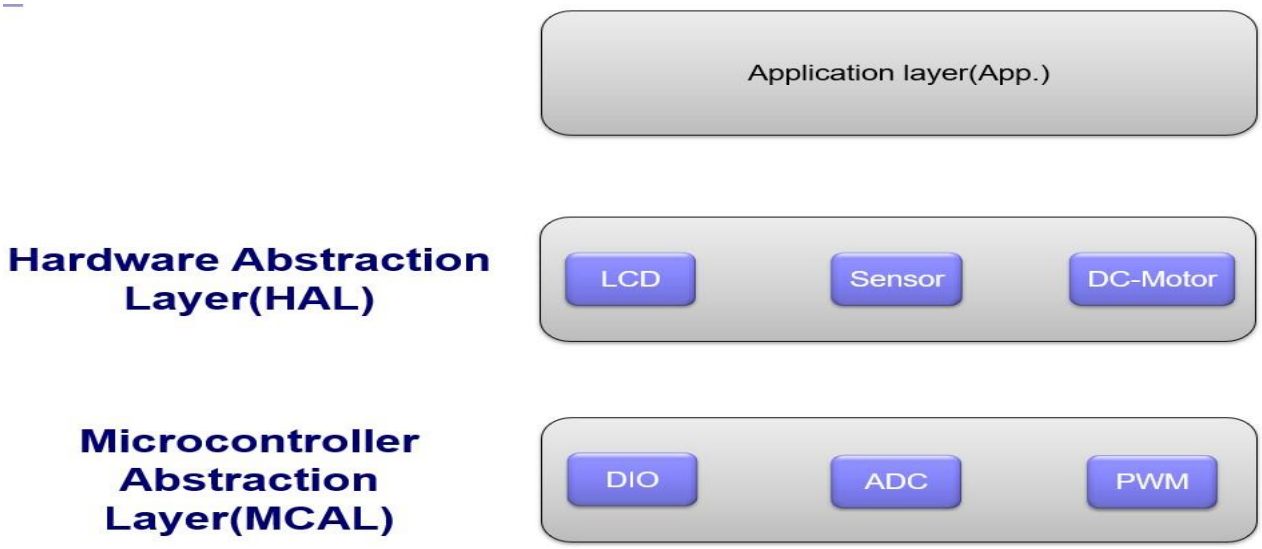
8. The LCD should display the temperature value and the fan state continuously like that:



9. Control the DC-Motor speed using PWM signal generated from **Timer0**.

10. Check this video: https://youtu.be/RFQGjcikfK4

11. The project should be design and implemented based on the layered architecture model as follow:



ADC Driver Requirements

1. Implement a **full ADC driver with the configuration technique based on the polling design**. You need to modify the **ADC_init** function implemented in the ADC session to take **a pointer to the configuration structure with type ADC_ConfigType.**
2. The function declaration should be**:**

   void ADC_init(const ADC_ConfigType * Config_Ptr)
3. The **ADC_ConfigType** structure should be declared like that:

   typedef struct{

   ADC_ReferenceVolatge ref_volt;

   ADC_Prescaler prescaler;

   }ADC_ConfigType;

   ADC_ ReferenceVolatge and **ADC_Prescaler** are types defined as **uint8** or **enum**. Check the target datasheet to gets the values.
4. ADC driver should configure to operate using the **internal reference voltage 2.56 voltage and prescaler F_CPU/8**.

GPIO Driver Requirements

1. Use the Same GPIO driver implemented in the course.

LCD Driver Requirements

2. Use the Same LCD driver implemented in the course with 8-bits data mode.
3. Connect the LCD control pins and 8-bits data pins as follow:

   RS → PD0

   RW → GROUND

   E → PD2

   Data Bus → all PORTC pins.

Temperature Sensor Driver Requirements

1. Use the same Temperature Sensor driver implemented in the course.
2. Connect the LM35 temperature sensor to **ADC channel 2**.

DC-Motor Driver Requirements

1. Implement a full DC-Motor Driver.
2. The DC Motor has 2 functions:
   a. **void DcMotor_Init(void)**
      - **Description**
        - The Function responsible for setup the direction for the two motor pins through the GPIO driver.
        - Stop at the DC-Motor at the beginning through the GPIO driver.
      - **Inputs: None**
      - **Return: None**
   b. **void DcMotor_Rotate(DcMotor_State state,uint8 speed)**
      - **Description:**
        - The function responsible for rotate the DC Motor CW/ or A-CW or stop the motor based on the state input state value.
        - Send the required duty cycle to the PWM driver based on the required speed value.
      - **Inputs:**
        - state: The required DC Motor state, it should be CW or A-CW or stop. **DcMotor_State** data type should be declared as **enum** or **uint8**.
        - speed: decimal value for the required motor speed, it should be from 0 → 100. For example, if the input is 50, The motor should rotate with 50% of its maximum speed.
      - **Return: None**

PWM Driver Requirements

1. No need to implement a full driver. Just use the same PWM Timer0 function implemented in the course. You will implement a full timer driver in the final project
2. The driver has one function:
   void PWM_Timer0_Start(uint8 duty_cycle)
   - **Description:**
     - The function responsible for trigger the Timer0 with the PWM Mode.
     - Setup the PWM mode with Non-Inverting.
     - Setup the prescaler with F_CPU/8.
     - Setup the compare value based on the required input duty cycle
     - Setup the direction for OC0 as output pin through the GPIO driver.
     - The generated PWM signal frequency will be 500Hz to control the DC Motor speed.
   - **Inputs:**
     - duty_cycle: The required duty cycle percentage of the generated PWM signal. Its value should be from 0 → 100
   - **Return: None**

Eng / Mohamed Morsy