**MICROPROCESSOR PROGRAMMING AND INTERFACING**  
**(CS-430)**

**Work by: Ali Raza**

**RAIN DETECTION AND CLASSIFICATION SYSTEM**

1. **OBJECTIVE**

To design and implement microprocessor/microcontroller-based system having interfaced an external memory device, an analog input device and a display output device.

1. **PROBLEM STATEMENT**

Students are required to design a microprocessor/microcontroller-based system using discrete ICs (trainer boards are not to be used). Students can form groups of maximum three members to work on the assigned task. There are two main tasks in this Complex Engineering Problem:

1. A software simulation on a circuit designing software (preferably Proteus) is to be designed for working demo of the project.
2. **ABSTRACT**

This technical report presents the design and implementation of a rain detection and classification system based on the ATmega328P microcontroller. The system utilizes an HW-83 analog rain sensor to monitor precipitation intensity, which is then interpreted and displayed using a 16x2 LCD module. A Pulse Width Modulation (PWM)-controlled LED provides real-time visual feedback based on rain severity. Additionally, the system records the latest rain classification in an external EEPROM (24LC512) for persistent storage. The entire project is developed using AVR Assembly language, focusing on low-power operation and cost-effectiveness for applications in smart agriculture, weather stations, and environmental monitoring.

Table of Contents

[**i.** **OBJECTIVE** 2](#_Toc203484741)

[**ii.** **PROBLEM STATEMENT** 2](#_Toc203484742)

[**iii.** **ABSTRACT** 2](#_Toc203484743)

[**1.** **INTRODUCTION** 5](#_Toc203484744)

[**2.** **SYSTEM OVERVIEW** 5](#_Toc203484745)

[**2.1.** **ATMEGA328P** 5](#_Toc203484746)

[**2.1.1.** **Key Features** 5](#_Toc203484747)

[**2.1.2.** **Architecture** 6](#_Toc203484748)

[**2.2.** **RAIN SENSOR (HW-83)** 8](#_Toc203484749)

[**2.2.1.** **Key Features** 8](#_Toc203484750)

[**2.2.2.** **Working Principle** 8](#_Toc203484751)

[**2.3.** **LM016L 16x2 LCD DISPLAY** 8](#_Toc203484752)

[**2.3.1.** **Key Features** 8](#_Toc203484753)

[**2.3.2.** **Function** 9](#_Toc203484754)

[**2.3.3.** **Advantages** 9](#_Toc203484755)

[**2.4.** **24C512 EEPROM** 9](#_Toc203484756)

[**2.4.1.** **Key Features** 9](#_Toc203484757)

[**2.4.2.** **Role** 9](#_Toc203484758)

[**2.4.3.** **Advantages** 10](#_Toc203484759)

[**2.5.** **PWM-CONTROLLED LED** 10](#_Toc203484760)

[**2.5.1.** **What is PWM?** 10](#_Toc203484761)

[**2.5.2.** **Role:** 10](#_Toc203484762)

[**3.** **CIRCUIT** 11](#_Toc203484763)

[**3.1.** **Working** 11](#_Toc203484764)

[**3.1.1.** **LCD Display** 11](#_Toc203484765)

[**3.1.2.** **LED Intensity Indication** 12](#_Toc203484766)

[**3.1.3.** **EEPROM Logging** 12](#_Toc203484767)

[**4.** **ASSEMBLY PROGRAM** 12](#_Toc203484768)

[**4.1.** **Initialization** 12](#_Toc203484769)

[**4.2.** **Start Routine** 13](#_Toc203484770)

[**4.3.** **Main Loop** 14](#_Toc203484771)

[**4.4.** **Rain Classification** 14](#_Toc203484772)

[**4.5.** **ADC Read (ADC\_Read)** 16](#_Toc203484773)

[**4.6.** **EEPROM Write (EEPROM\_WriteFull)** 16](#_Toc203484774)

[**4.7.** **Print EEPROM Content (Print\_EEPROM)** 17](#_Toc203484775)

[**4.8.** **I²C (TWI) Subroutines** 18](#_Toc203484776)

[**4.9.** **LCD Subroutines** 19](#_Toc203484777)

[**4.10.** **Delays** 19](#_Toc203484778)

[**5.** **SIMULATION** 20](#_Toc203484779)

[**5.1.** **Heavy Rain (**At ADC>=700**)** 20](#_Toc203484780)

[**5.2.** **Moderate Rain (**At 512<=ADC<=699**)** 20](#_Toc203484781)

[**5.3.** **Light Rain (**At 256<=ADC<=511**)** 20](#_Toc203484782)

[**5.4.** **No Rain (**At ADC<=255**)** 20](#_Toc203484783)

[**6.** **CONCLUSION** 20](#_Toc203484784)

1. **INTRODUCTION**

Rain monitoring is a crucial aspect of various modern systems, including precision agriculture, automated irrigation, and environmental forecasting. Traditional systems often require high computational resources or are not optimized for embedded environments. This project demonstrates a compact and efficient rain detection and notification system built around the ATmega328P microcontroller, using simple analog input from an HW-83 sensor. The project is entirely written in AVR Assembly to provide deep-level hardware control, allowing optimized resource use on an 8-bit microcontroller.

1. **SYSTEM OVERVIEW**

The rain detection system consists of the following hardware components:

* 1. **ATMEGA328P**

The ATmega328P is an 8-bit microcontroller from Atmel’s AVR family, widely recognized for its use in Arduino boards, particularly the Arduino Uno. In this rain detector system project, the ATmega328P serves as the central processing unit responsible for interfacing with sensors, processing input signals, and controlling outputs such as buzzers or LEDs to alert users when rain is detected.

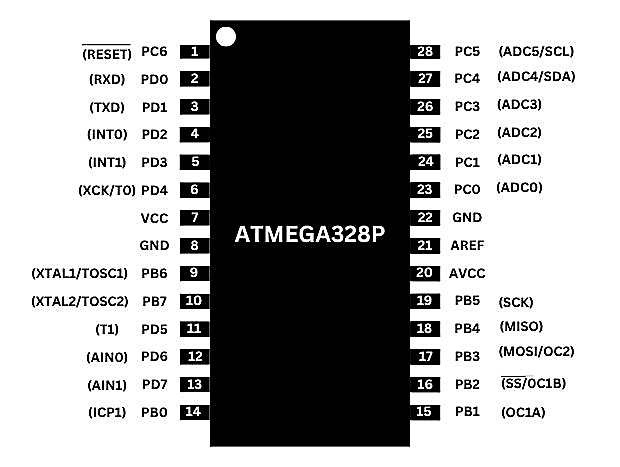
* + 1. **Key Features**
* **Architecture**: 8-bit RISC (Reduced Instruction Set Computing)
* **Operating Voltage**: 1.8V – 5.5V
* **Flash Memory**: 32 KB (with 0.5 KB used for bootloader)
* **SRAM**: 2 KB
* **EEPROM**: 1 KB
* **Clock Speed**: Up to 20 MHz
* **Digital I/O Pins**: 23
* **Analog Input Pins**: 6 (10-bit ADC)

Figure - Atmega328P Pin Configuration

The ATmega328P was chosen for its low power consumption, ease of programming, and compatibility with a wide range of input/output components. In the rain detection system, it reads the signal from a rain sensor module (e.g., analog or digital output), processes the input using its built-in ADC or digital pin logic, and triggers an appropriate response based on predefined conditions. For example, when rain is detected, the microcontroller can activate an alert or send a signal to a connected module, such as a GSM module for remote notification.

* + 1. **Architecture**

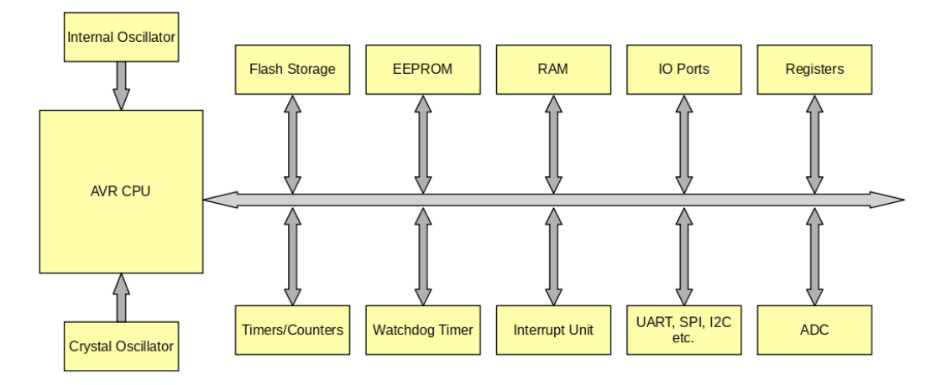
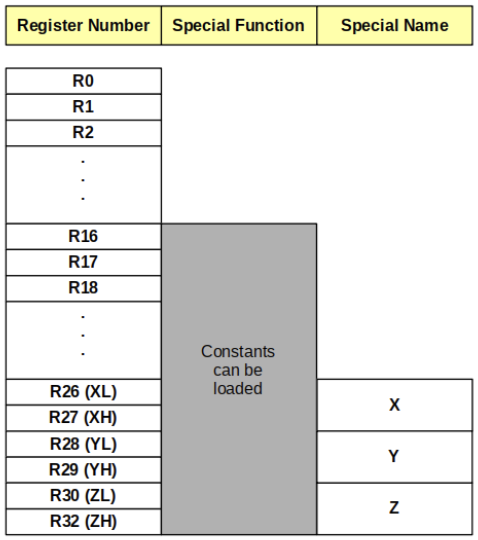
The ATmega328P uses an 8-bit Reduced Instruction Set Computing (RISC) architecture. It executes most instructions in a single clock cycle, which enhances performance and reduces power consumption.

Figure - Atmega328P Architecture

* **Registers**

The ATmega328P includes a total of 32 general-purpose 8-bit registers (R0 to R31). These registers are directly connected to the ALU (Arithmetic Logic Unit), enabling high-speed data processing.

* + **Working Registers (R0–R31):** These allow direct operations like addition, subtraction, logic, and shift without memory access.
  + **Pointer Registers:** It also has three 16-bit pointer registers (X, Y, Z) formed using register pairs:

X = R27:R26  
Y = R29:R28  
Z = R31:R30

These are used for indirect addressing, especially during memory operations or string manipulation.

* + **Status Register (SREG):** Holds flags like Zero (Z), Carry (C), Negative (N), Overflow (V), and Global Interrupt Enable (I), which control program flow based on arithmetic or logic outcomes.

Figure - Atmega328P Internal Registers

* **I/O Ports**

The microcontroller has 23 programmable I/O lines across three ports:

* + PORTB (PB0–PB7): 8 bits, used for general I/O, PWM outputs, and SPI communication.
  + PORTC (PC0–PC6): 7 bits, includes analog inputs (ADC0–ADC5) and I²C pins (SDA and SCL).
  + PORTD (PD0–PD7): 8 bits, includes UART (TX, RX), interrupts, and PWM outputs.

Each pin can be configured independently as input or output using the DDRx (Data Direction Register) and manipulated using PORTx and PINx registers.

* **Timers and Counters**

The ATmega328P contains three timers, each with unique features:

* + Timer0 (8-bit): Supports PWM (Fast and Phase Correct). Commonly used for time delays and periodic interrupts
  + Timer1 (16-bit): Offers high-resolution timing. Used for precise event timing, input capture, and servo control
  + Timer2 (8-bit): Used for simple timing tasks or generating additional PWM signals.These timers are controlled using registers like TCCRnA, TCCRnB, TCNTn, OCRnA/B, and TIMSKn.
* **Analog-To-Digital Convertor (ADC)**

The ATmega328P features a 10-bit ADC capable of converting analog signals into digital values with a resolution of 0–1023.

* + Channels: Up to 6 analog input channels (ADC0–ADC5) on PORTC
  + Voltage Reference: Configurable (AVcc, Internal 1.1V, or External AREF)
  + Prescaler: Selectable to control ADC clock speed for accuracy
  + Control Registers: ADMUX selects the channel and reference, ADCSRA enables ADC and controls conversion
* **Communication Interfaces**

The ATmega328P supports **three main serial communication protocols**:

* + **USART (Universal Synchronous/Asynchronous Receiver/Transmitter):** Used for serial communication with PCs or other devices. Supports baud rate generation and interrupt-based transmission.
  + **I²C (Inter-Integrated Circuit or TWI – Two-Wire Interface):** Master/slave serial bus using SDA and SCL lines (PC4 and PC5). Used to communicate with EEPROMs (like the 24C512), sensors, and other peripherals.
  + **SPI (Serial Peripheral Interface):** Full-duplex, high-speed communication using MOSI, MISO, SCK, and SS pins. Commonly used for memory cards, displays, or other high-speed peripherals.

These interfaces make the ATmega328P highly versatile and suitable for complex embedded systems requiring multiple peripherals.

* 1. **RAIN SENSOR (HW-83)**

The HW-83 Rain Sensor is a simple and effective module designed to detect water droplets or rainfall. It is commonly used in weather monitoring systems, automatic wiper systems, and home automation projects. In this project, the HW-83 serves as the primary sensor responsible for detecting rain and sending corresponding signals to the microcontroller (ATmega328P).

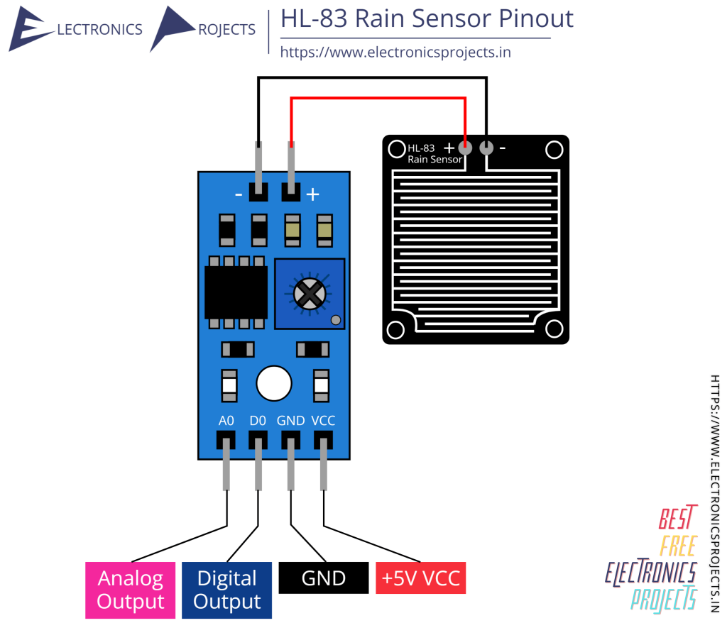
* + 1. **Key Features**
* **Operating Voltage:** 3.3V – 5V
* **Output Types:** Analog and Digital
* **Detection Principle:** Water conductivity
* **Corrosion-resistant design:** For outdoor use
* **Adjustable Sensitivity:** Via onboard potentiometer
* **Dual-Part Structure:**
  + Sensor Plate: Detects rain through water conductivity
  + Control Board: Processes signals and provides outputs
    1. **Working Principle**

Figure - HW-83 Rain Sensor

The HW-83 rain sensor consists of two main parts: a sensor plate (PCB with exposed conductive lines) and a signal processing board. When raindrops fall on the sensor plate, they create a conductive path between the traces, altering the resistance. This change is detected and converted into both analog and digital signals.

* The analog output provides a variable voltage based on the amount of water on the sensor.
* The digital output is triggered when the water level crosses a set threshold, which can be adjusted using the onboard potentiometer.

In the rain detector system, the digital output is connected to one of the ATmega328P's digital input pins. When rain is detected, the microcontroller receives a signal, depending on the rain level, and then takes appropriate action, such as activating an alarm or notification system.

The HW-83 is favoured for its ease of use, reliability, and compatibility with microcontrollers, making it a practical choice for this project.

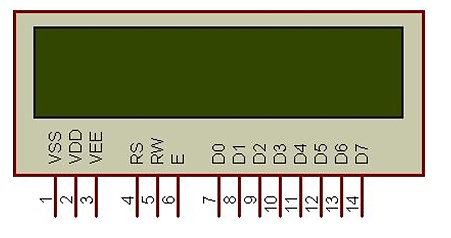
* 1. **LM016L 16x2 LCD DISPLAY**

The LM016L 16x2 LCD Display is a widely used alphanumeric liquid crystal display that shows up to 16 characters per line on 2 lines, making it ideal for displaying messages and sensor status in embedded systems like the rain detector.

* + 1. **Key Features**
* **Display Format**: 16 characters × 2 lines
* **Interface**: Parallel (4-bit or 8-bit mode)
* **Controller**: Based on the HD44780 LCD controller
* **Operating Voltage**: Typically, 5V.
* **Backlight**: LED backlight for readability in low light

Figure - LM016L LCD Display

* **Adjustable Contrast**: Using an external potentiometer
  + 1. **Function**

In this project, the LM016L is used to provide real-time **visual feedback** to the user. It displays system messages such as:

* “No Rain” at
* “Light Rain” at
* “Moderate Rain” at
* “Heavy Rain” at

The display is connected to the ATmega328P microcontroller, via **8-bit data**. Control lines like **RS (Register Select), RW (Read/Write), and E (Enable)** are used to manage communication between the microcontroller and the display.

Figure - LM016L LCD pinout

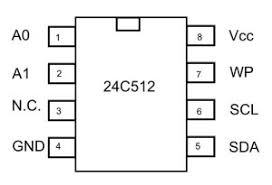
* + 1. **Advantages**
* Clear, user-friendly status messages
* Low power consumption
* Simple interfacing with the ATmega328P
* Reliable for outdoor monitoring systems with basic enclosure protection

The LM016L enhances the rain detector system by providing an interactive interface for users to monitor conditions without requiring a serial monitor or external computer.

* 1. **24C512 EEPROM**

The 24C512 is a 512-Kilobit (64KB) serial Electrically Erasable Programmable Read-Only Memory (EEPROM) that communicates using the I²C (Inter-Integrated Circuit) protocol. It provides non-volatile storage, meaning the data remains intact even when the system is powered off. In this project, the 24C512 is used to store the latest rain classification data recorded by the ATmega328P microcontroller.

* + 1. **Key Features**



* **Memory Size:** 512 Kbits (65,536 × 8 bits = 64 KB)
* **Interface:** I²C (Two-wire: SDA and SCL)
* **Operating Voltage:** Typically 1.7V to 5.5V
* **Endurance:** Up to 1 million write/erase cycles
* **Data Retention:** More than 100 years
* **Write Protection:** Via software or external pin
* **Page Write Size:** 128 bytes per page

Figure - 24LC512 EEPROM pin configuration

* **Packages Available:** 8-pin DIP, SOIC, TSSOP
  + 1. **Role**

The 24C512 EEPROM is used to store the current rain category (e.g., "No Rain", "Light Rain", "Moderate Rain", or "Heavy Rain"). This allows the system to retain the last known rain status even after a power cycle. The ATmega328P communicates with the EEPROM via the I²C protocol, writing data to the memory and retrieving it during system startup to display on the LCD.

* + 1. **Advantages**
* **Non-volatile Memory:** Retains data after power loss
* **High Capacity:** Suitable for storing multiple event logs or configuration data
* **Efficient I²C Communication:** Conserves I/O pins and supports multiple devices on the same bus
* **Easy Integration:** Compatible with most microcontrollers including AVR-based systems like ATmega328P

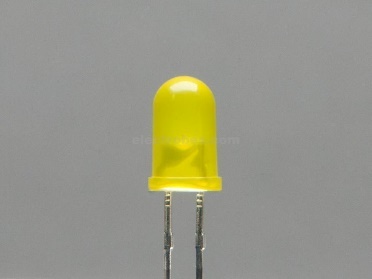
By incorporating the 24C512 EEPROM, the rain detection system enhances its reliability and functionality by preserving essential environmental data between sessions, enabling potential long-term data logging or pattern analysis.

* 1. **PWM-CONTROLLED LED**

In the rain detector system, a PWM-controlled LED is used to provide a visual indication of rain intensity or system status. PWM, or Pulse Width Modulation, is a technique that allows control over the brightness of an LED by varying the duty cycle of a digital signal.

* + 1. **What is PWM?**

Pulse Width Modulation is a method used in digital systems to simulate analog output. By rapidly switching a digital pin ON and OFF at a specific frequency and varying the ON-time (duty cycle), the average power delivered to the LED is changed, which in turn affects its brightness.

* + 1. **Role:**

The PWM-controlled LED can be used to:

* **Indicate rain intensity**: Higher brightness corresponds to heavier rain.
* **Display status levels**: Dim for "No Rain", medium for "Light Rain", and bright for "Heavy Rain".
* **Create visual alerts** in noisy environments where a buzzer might not be effective.

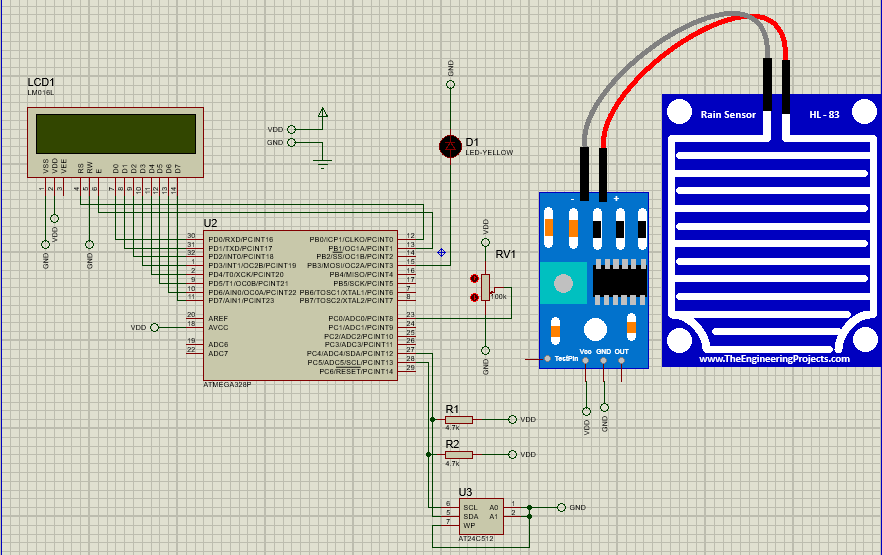
Figure - Light Emitting Diode

The ATmega328P microcontroller has built-in PWM capabilities on several of its digital pins (e.g., D3, D5, D6, D9, D10, D11 on Arduino Uno), which are used to control the LED brightness dynamically using software.

The system workflow involves reading analog signals from the rain sensor, interpreting the signal via the ADC, and classifying the rain intensity into one of four categories. The classification is stored in EEPROM and simultaneously shown on the LCD. A PWM-driven LED changes brightness to reflect increasing rain intensity in real time.

1. **CIRCUIT**

Figure - Circuit Diagram

* 1. **Working**

**To simulate the analog output of Rain sensor in the proteus software a potentiometer is used because the analog pin does not exist for the Rain sensor in proteus.**

The rain detection and classification system operate by continuously monitoring the output of the HL-83 rain sensor, which is interfaced with the ADC0 (PC0) pin of the ATmega328P microcontroller. The sensor provides an analog voltage proportional to the amount of moisture detected. This signal is digitized using the internal Analog-to-Digital Converter (ADC) of the microcontroller.

The digitized value is then evaluated against predefined threshold levels to determine the intensity of rainfall. Based on these thresholds, the system classifies the rainfall into one of four categories:

* No Rain
* Light Rain
* Moderate Rain
* Heavy Rain

Each category triggers a coordinated response from the system comprising three key actions:

### **LCD Display**

A classification message (e.g., "Light Rain") is displayed on a 16×2 alphanumeric LCD (LM016L), which is controlled in 8-bit mode using Port D for data lines and Port B for the RS (Register Select) and E (Enable) control lines.

### **LED Intensity Indication**

An LED connected to PB3 (OC2A) is driven using PWM (Pulse Width Modulation) via Timer2. The duty cycle is adjusted according to the rainfall category, resulting in different brightness levels to visually indicate rain intensity — off for no rain and fully bright for heavy rain.

### **EEPROM Logging**

The classification string is stored in an external I2C EEPROM (24LC256) to retain the last known state even after a power cycle. The data is written and read via the TWI (Two-Wire Interface) using the SDA and SCL lines connected to the microcontroller.

To optimize EEPROM usage and minimize unnecessary writes, the system compares the current rainfall category with the last stored one. If a change is detected, the new classification is written to the EEPROM and updated on the LCD. Otherwise, the system retains the previous state.

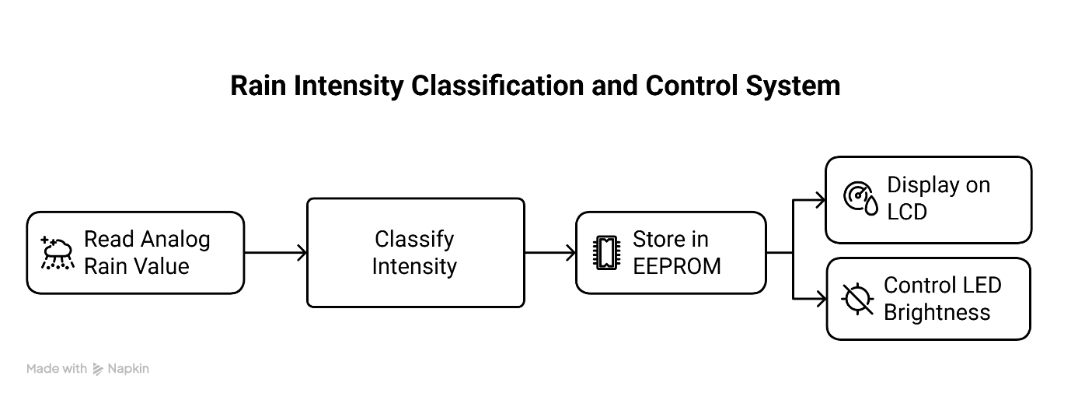
This setup allows for real-time rainfall monitoring with persistent logging and intuitive visual and textual feedback.

Figure - Work Flow Block Diagram

1. **ASSEMBLY PROGRAM**

Following section explains the AVR assembly code used to implement a rain detection and classification system.

* 1. **Initialization**
* **Includes and Constants:**

The .include directive loads register definitions specific to the ATmega328P. Constants are defined for LCD control lines (RS, E) and a buffer size (EEPROM\_STRING\_LIMIT = 16).

* **Data Section (.dseg):**

Allocates memory:

* + Buffer: Temporary storage for the rain category string.
  + LastRainCategory: Holds the previously detected category to avoid redundant updates.

1. .include "m328Pdef.inc"
2. ; ===== Constants =====
3. .equ RS = 0
4. .equ E = 1
5. .equ EEPROM\_STRING\_LIMIT = 16 ; Max string length to store in EEPROM
6. ; ===== Data Section =====
7. .dseg
8. Buffer: .byte EEPROM\_STRING\_LIMIT ; Buffer for storing current rain string
9. LastRainCategory: .byte 1 ; Holds last rain classification
   1. **Start Routine**

The Start: label initializes all peripherals:

* **PWM (LED Brightness):** Timer2 is configured in Fast PWM mode on PB3 to control LED intensity based on rain level.
* **ADC (Analog Input):** PC0 is set as an input to receive data from the rain sensor.
* **LCD Setup:** Initializes LCD in 8-bit mode with control via PORTB and data via PORTD.
* **TWI (I²C for EEPROM):** Initializes communication with the external EEPROM.
* **ADC Setup:** Enables ADC and selects ADC0 as input.
* **First Read:** Reads sensor data, classifies rain level, stores the result in EEPROM, and prints it on the LCD.

1. ; ===== Code Section =====
2. .cseg
3. .org 0x0000
4. rjmp Start ; Reset vector
5. ; ===== String Constants =====
6. .org 0x0200
7. LightRain: .db "Light Rain ", 0
8. ModRain: .db "Moderate Rain ", 0
9. HeavyRain: .db "Heavy Rain ", 0
10. NoRain: .db "No Rain ", 0
11. ; ===== Start Routine =====
12. Start:
13. ; --- PWM Init on PB3 (OC2A) for LED intensity ---
14. sbi DDRB, PB3 ; PB3 as output
15. ldi r16, (1<<COM2A1)|(1<<WGM21)|(1<<WGM20) ;set r16 to 1000 0011 or 0x83h
16. sts TCCR2A, r16 ; set the PWM to fast, non-inverting mode
17. ldi r16, (1<<CS21) ; Prescaler = 8
18. sts TCCR2B, r16 ; CS21 = bit 1 (0x02h)
19. ; --- ADC Init: PC0 as input ---
20. ldi r16, 0x00
21. out DDRC, r16 ; All PCx as input
22. out PORTC, r16 ; Disable pull-ups
23. ; --- LCD Init ---
24. ldi r16, 0xFF
25. out DDRD, r16 ; LCD data pins (D0–D7) as output (0xFF)
26. in r16, DDRB ; load prev state of DDRB to not overwrite
27. ori r16, (1<<RS)|(1<<E)
28. out DDRB, r16 ; RS and E as output
29. ldi r16, 0
30. out PORTB, r16 ; Clear control lines after initialization
31. rcall LongDelay
32. rcall LongDelay
33. ldi r16, 0x38 ; 8-bit mode, 2 lines, 5x8 dot font
34. rcall LCD\_Command
35. ldi r16, 0x0C ; Display ON, cursor OFF
36. rcall LCD\_Command
37. ldi r16, 0x01 ; Clear display
38. rcall LCD\_Command
39. rcall LongDelay
40. ldi r16, 0x06 ; Entry mode set
41. rcall LCD\_Command
42. ; --- I2C Init for EEPROM ---
43. rcall TWI\_Init
44. ; --- ADC Setup ---
45. ldi r16, (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
46. sts ADCSRA, r16 ; Prescaler set to 128
47. ldi r16, 0 ; ADC0 channel
48. sts ADMUX, r16
49. ; --- Initial Read, Classify, Store & Display ---
50. rcall ADC\_Read
51. rcall ClassifyRain
52. rcall EEPROM\_WriteFull
53. rcall LCD\_Clear
54. rcall Print\_EEPROM
    1. **Main Loop**

* Continuously reads sensor data using ADC\_Read.
* ClassifyRain determines the rain level.
* If the rain category changes, it updates LastRainCategory, writes the new string to EEPROM, and updates the LCD display.
* If no change is detected, it skips the update to save EEPROM write cycles.

1. ; ===== Main Loop =====
2. Loop:
3. rcall ADC\_Read
4. rcall ClassifyRain ; Updates r17 with new category
5. lds r18, LastRainCategory
6. cp r17, r18
7. breq SkipUpdate ; No change detected
8. sts LastRainCategory, r17 ; Update stored category
9. rcall EEPROM\_WriteFull
10. rcall LCD\_Clear
11. rcall Print\_EEPROM
12. SkipUpdate:
13. rcall LongDelay
14. rjmp Loop
    1. **Rain Classification**

Compares ADC result against thresholds, Each level:

* Sets an ID in r17
* Loads a corresponding string (NoRain, LightRain, etc.)
* Sets PWM duty cycle for LED brightness.
* Copies the string to Buffer.

1. ; ===== Classify Rain Based on ADC Reading =====
2. ; Sets r17 to category ID and stores string in Buffer
3. ClassifyRain:
4. ; Check thresholds and assign message + PWM
5. ldi r20, low(255)
6. ldi r21, high(255)
7. cp r24, r20
8. cpc r25, r21 ; Compare the ADC result with threshold
9. brlo LabelNoRain
10. ldi r20, low(512)
11. ldi r21, high(512)
12. cp r24, r20
13. cpc r25, r21
14. brlo LabelLight
15. ldi r20, low(700)
16. ldi r21, high(700)
17. cp r24, r20
18. cpc r25, r21
19. brlo LabelModerate
20. ; --- Heavy Rain ---
21. ldi r17, 3
22. ldi ZH, high(HeavyRain << 1)
23. ldi ZL, low(HeavyRain << 1)
24. ldi r16, 255
25. sts OCR2A, r16
26. rjmp CopyMsg
27. LabelModerate:
28. ldi r17, 2
29. ldi ZH, high(ModRain << 1)
30. ldi ZL, low(ModRain << 1)
31. ldi r16, 170
32. sts OCR2A, r16
33. rjmp CopyMsg
34. LabelLight:
35. ldi r17, 1
36. ldi ZH, high(LightRain << 1)
37. ldi ZL, low(LightRain << 1)
38. ldi r16, 85
39. sts OCR2A, r16
40. rjmp CopyMsg
41. LabelNoRain:
42. ldi r17, 0
43. ldi ZH, high(NoRain << 1)
44. ldi ZL, low(NoRain << 1)
45. ldi r16, 0
46. sts OCR2A, r16
47. CopyMsg:
48. ; Copy string from Flash to SRAM buffer
49. ldi YH, high(Buffer) ; Load high byte of destination buffer addr
50. ldi YL, low(Buffer) ; Load low byte
51. ldi r18, EEPROM\_STRING\_LIMIT
52. CopyMsgLoop:
53. lpm r16, Z+ ; Load byte from flash pointed to by Z, then inc
54. st Y+, r16 ; Increment addr in Y after storing r16 in SRAM
55. dec r18
56. cpi r16, 0
57. breq CopyMsgDone
58. brne CopyMsgLoop
59. CopyMsgDone:
60. Ret
    1. **ADC Read (ADC\_Read)**

Performs a 10-bit ADC conversion:

* Configures reference voltage and starts conversion
* Waits for conversion to complete
* Stores result in r24:r25

1. ; ===== ADC Read into r24:r25 =====
2. ADC\_Read:
3. ldi r16, (1<<REFS0) ; set AVCC as Ref Voltage
4. sts ADMUX, r16
5. ldi r16, (1<<ADEN)|(1<<ADSC)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
6. sts ADCSRA, r16 ; Enables, Start Conversation
7. Wait\_ADC:
8. lds r16, ADCSRA
9. sbrc r16, ADSC ; No Skip if ADSC bit is set (ADC busy)
10. rjmp Wait\_ADC
11. lds r24, ADCL ; Lower byte of ADC PC0
12. lds r25, ADCH ; Higher byte
13. Ret
    1. **EEPROM Write (EEPROM\_WriteFull)**

Sends the entire buffer over I²C:

* Uses standard EEPROM address (0xA0)
* Sends 16-byte string from Buffer using page write
* Waits until the write completes using polling

1. ; ===== EEPROM Write: Write full buffer over I2C =====
2. EEPROM\_WriteFull:
3. rcall TWI\_Start
4. ldi r24, 0xA0 ; Device write addr
5. rcall TWI\_Write
6. ldi r24, 0 ; Word addr high
7. rcall TWI\_Write
8. ldi r24, 0 ; Word addr low
9. rcall TWI\_Write
10. ldi YH, high(Buffer)
11. ldi YL, low(Buffer)
12. ldi r18, EEPROM\_STRING\_LIMIT
13. WriteLoop:
14. ld r24, Y+
15. rcall TWI\_Write
16. dec r18
17. brne WriteLoop
18. rcall TWI\_Stop
19. rcall EEPROM\_Wait
20. Ret
21. ; ===== Wait for EEPROM Write Completion =====
22. EEPROM\_Wait:
23. rcall TWI\_Start
24. ldi r24, 0xA0
25. rcall TWI\_Write
26. brne EEPROM\_Wait ; ACK polling
27. rcall TWI\_Stop
28. ret
    1. **Print EEPROM Content (Print\_EEPROM)**

Reads back the 16-byte string from EEPROM and displays it on the LCD:

* Reads data using TWI\_Read\_ACK/NACK
* Sends each character to the LCD

1. ; ===== Print EEPROM String to LCD =====
2. Print\_EEPROM:
3. rcall TWI\_Start ; EEPROM read
4. ldi r24, 0xA0
5. rcall TWI\_Write
6. ldi r24, 0
7. rcall TWI\_Write
8. ldi r24, 0
9. rcall TWI\_Write
10. rcall TWI\_Start
11. ldi r24, 0xA1 ; EEPROM read
12. rcall TWI\_Write
13. ldi r19, 0,
14. PrintLoop:
15. cpi r19, EEPROM\_STRING\_LIMIT - 1
16. breq LastByte
17. rcall TWI\_Read\_ACK
18. rjmp ShowByte
19. LastByte:
20. rcall TWI\_Read\_NACK
21. ShowByte:
22. mov r16, r24
23. cpi r16, 0
24. breq PrintEnd
25. rcall LCD\_Data
26. inc r19
27. cpi r19, EEPROM\_STRING\_LIMIT
28. brne PrintLoop
29. PrintEnd:
30. rcall TWI\_Stop
31. Ret
    1. **I²C (TWI) Subroutines**

Handles I²C communication:

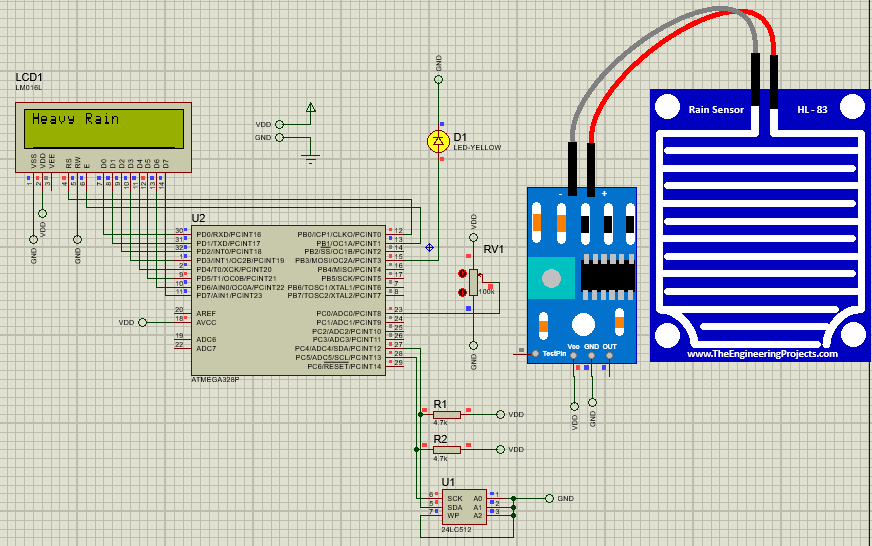
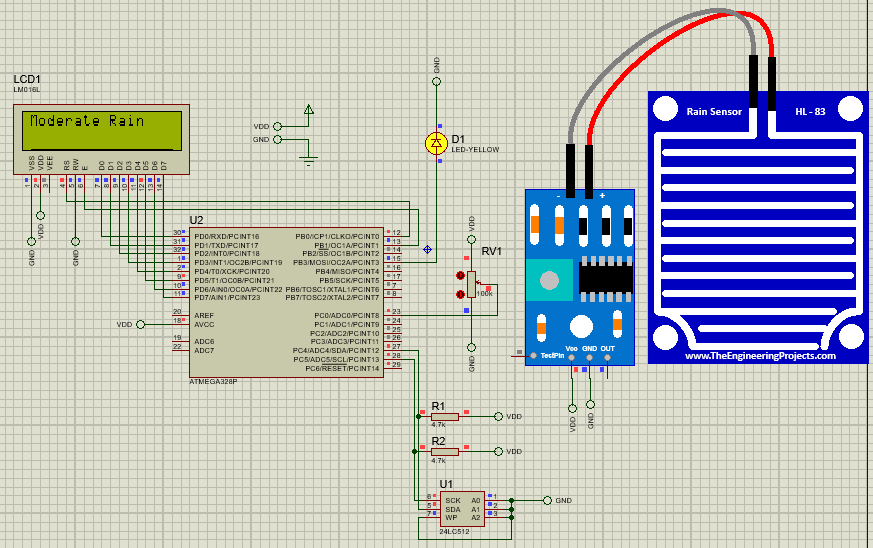
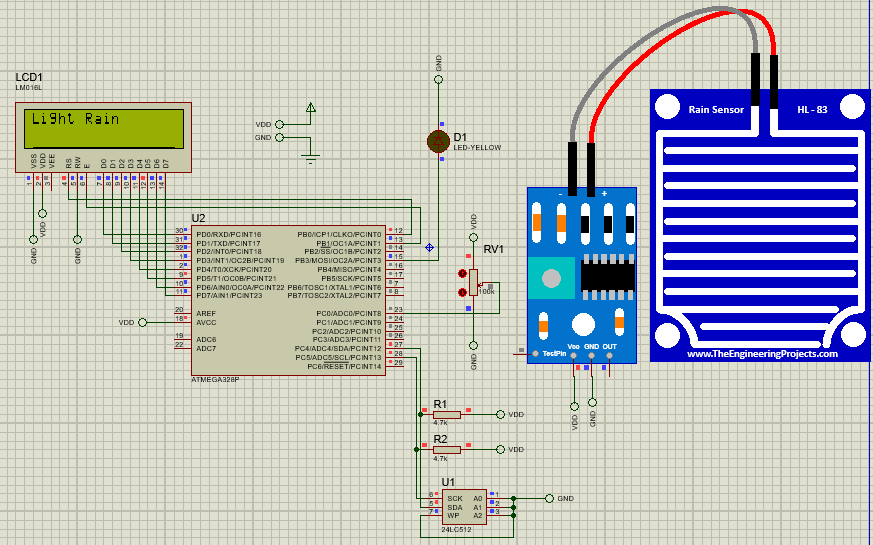
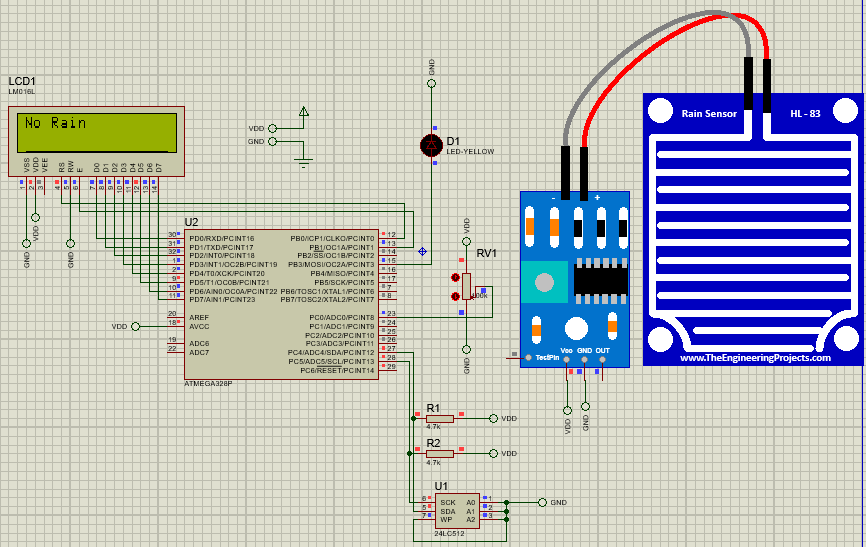
* TWI\_Start, TWI\_Stop: Begin and end transmission
* TWI\_Write: Write a byte
* TWI\_Read\_ACK/NACK: Read byte with/without acknowledgment

1. TWI\_Init:
2. ldi r16, 0
3. sts TWSR, r16
4. ldi r16, 0x20 ; TWBR = 32 for ~100kHz at 8MHz
5. sts TWBR, r16
6. ret
7. TWI\_Start:
8. ldi r16, (1<<TWSTA)|(1<<TWEN)|(1<<TWINT) ; Start Condition TWSTA
9. sts TWCR, r16
10. WaitStart:
11. lds r16, TWCR
12. sbrs r16, TWINT ; TWINT not yet set, continue waiting
13. rjmp WaitStart
14. ret
15. TWI\_Stop:
16. ldi r16, (1<<TWSTO)|(1<<TWEN)|(1<<TWINT)
17. sts TWCR, r16
18. ret
19. TWI\_Write:
20. sts TWDR, r24 ; Lower byte to be transferred
21. ldi r16, (1<<TWEN)|(1<<TWINT)
22. sts TWCR, r16
23. WaitWrite:
24. lds r16, TWCR
25. sbrs r16, TWINT ; TWINT not yet set, continue waiting
26. rjmp WaitWrite
27. ret
28. TWI\_Read\_ACK:
29. ldi r16, (1<<TWEN)|(1<<TWINT)|(1<<TWEA) ; Clear Interrupt to start trans.
30. sts TWCR, r16
31. WaitReadACK: ; Works same as WaitStart
32. lds r16, TWCR
33. sbrs r16, TWINT
34. rjmp WaitReadACK
35. lds r24, TWDR
36. ret
37. TWI\_Read\_NACK:
38. ldi r16, (1<<TWEN)|(1<<TWINT) ; TWEA not set, NACK after receiving
39. sts TWCR, r16
40. WaitReadNACK:
41. lds r16, TWCR
42. sbrs r16, TWINT
43. rjmp WaitReadNACK
44. lds r24, TWDR
45. Ret
    1. **LCD Subroutines**

* LCD\_Command and LCD\_Data: Send commands and characters
* LCD\_Clear: Clears the display
* LCD\_Write: Writes one byte using PORTD and toggles E

1. ; ===== LCD Subroutines =====
2. LCD\_Command:
3. cbi PORTB, RS ; RS=0 means instruction input
4. rjmp LCD\_Write
5. LCD\_Data:
6. sbi PORTB, RS
7. rjmp LCD\_Write
8. LCD\_Write:
9. out PORTD, r16
10. sbi PORTB, E
11. rcall ShortDelay
12. cbi PORTB, E
13. rcall ShortDelay
14. ret
15. LCD\_Clear:
16. ldi r16, 0x01 ; Clear command
17. rcall LCD\_Command
18. rcall LongDelay
19. Ret
    1. **Delays**

* ShortDelay and LongDelay: Provide timing control using nested loops, important for LCD timing and visual feedback

1. ; ===== Delay Subroutines =====
2. ShortDelay:
3. ldi r20, 100
4. LoopS:
5. dec r20
6. brne LoopS
7. ret
8. LongDelay:
9. ldi r21, 10
10. LoopL:
11. rcall ShortDelay
12. dec r21
13. brne LoopL
14. Ret
15. **SIMULATION**
    1. **Heavy Rain (**At ADC>=700**)**
    2. **Moderate Rain (**At 512<=ADC<=699**)**
    3. **Light Rain (**At 256<=ADC<=511**)**
    4. **No Rain (**At ADC<=255**)**
16. **CONCLUSION**

The rain detection and classification system presented in this report demonstrates an effective integration of analog sensing, signal processing, data display, and non-volatile storage using the ATmega328P microcontroller. By leveraging AVR Assembly language, the design achieves efficient, low-level control over hardware resources, ensuring low power consumption and cost-efficiency. The use of a PWM-controlled LED for visual indication, an LCD for real-time feedback, and external EEPROM for data persistence highlights the system's practicality and robustness. This solution is well-suited for deployment in smart agriculture, weather monitoring, and other environmental sensing applications where reliability and simplicity are critical. Future enhancements could include wireless data transmission or solar power integration to further extend the system’s applicability in remote and resource-constrained environments.