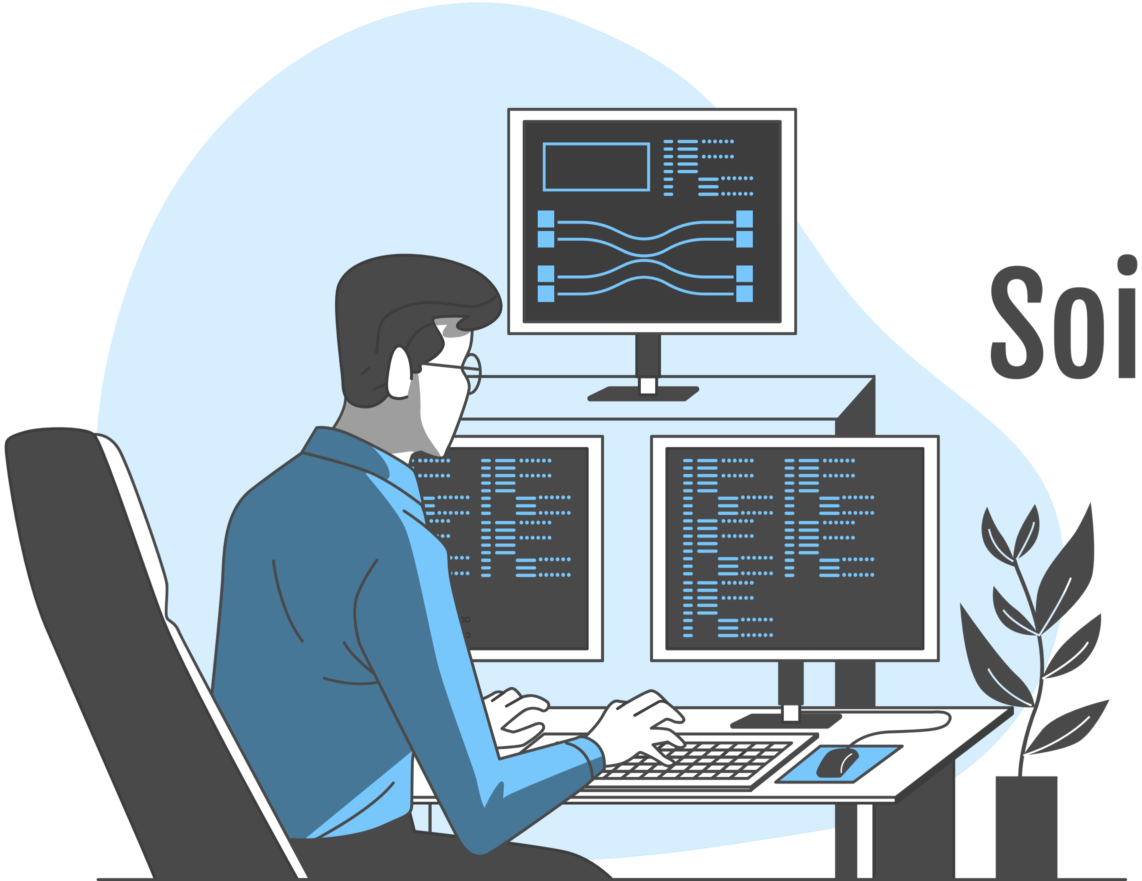


Soil Irrigation System

Optimizing Crop Growth

Team Member

01 Ahmed Hussein Badr Rashad

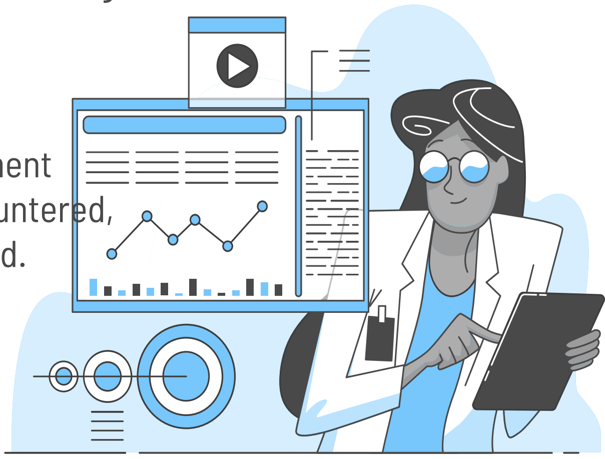
02 Ahmed Tarek Abdul Sattar Ahmed

03 Ahmed Ali Fahmy Muhammad

04 Abdul Rahman Helal Saleh Abdul Aziz

05 Abdullah Eid El-Sayed Ammar

Table of Contents



01
Introduction

Introduction

In agriculture, managing soil moisture efficiently is vital. Our project offers an automated solution using embedded systems. It utilizes the **ATmega32** microcontroller alongside sensors and pumps to monitor and regulate soil moisture levels accurately.

Our system operates autonomously or on user-defined intervals, **optimizing water usage**. Through **UART communication** and **Python scripting**, it **provides real-time data monitoring and visualization**, aiding farmers in decision-making for better crop growth and resource management.

02

Project Overview

Project Overview

This project combines various hardware components to create an efficient and reliable irrigation system. The **core** of the system is the **ATmega32** microcontroller, which coordinates the operation of **soil moisture sensors**, a **water pump**, an **LCD** display, a **keypad**, a **relay**, and a **power supply**.

The system operates in **two modes**: **autonomous** and **user-defined**. In autonomous mode, it continuously monitors soil moisture levels using the ADC (Analog-to-Digital Converter) and activates the water pump when moisture falls below 50%. In user-defined mode, the user can set a time interval via the keypad, and the system will assess soil moisture and activate irrigation based on the set interval, minimizing sensor wear.

Project Overview

Additionally, the project utilizes **UART** (Universal Asynchronous Receiver-Transmitter) communication to **send soil moisture data to a laptop**. **Python scripts capture** and **visualize** this data, providing users with insights into soil moisture trends over time. This automated solution aims to improve water efficiency and crop health by ensuring precise irrigation.

03

Hardware Components

Hardware Components

01 **ATmega32 Microcontroller**

brain of the system, managing all inputs and outputs, processing data from sensors

02 **Soil Moisture Sensors**

Measure the moisture level in the soil

03 **Water Pump**

Responsible for irrigating the soil

04 **LCD Display**

Provides a user interface to display system status

05 **Keypad**

Allows users to interact with the system

06 **Power Supply**

Powers all components of the system

04

Operating Modes

Operating Modes(Autonomous Mode)

Function: In this mode, the system autonomously monitors soil moisture levels and activates irrigation as needed.

Operation: The soil moisture sensors continuously provide analog data to the ATmega32 microcontroller's ADC (Analog-to-Digital Converter). When the moisture level falls below a predefined threshold (e.g., 50%), the microcontroller activates the relay to turn on the water pump. The pump irrigates the soil until the moisture level returns to an acceptable range.

Benefit: This mode ensures that plants receive adequate water supply without any manual intervention, maintaining optimal soil moisture levels for healthy plant growth.

Operating Modes(User-Defined Interval Mode)

Function: This mode allows users to set specific irrigation intervals, minimizing sensor wear and managing water usage efficiently.

Operation: Users can input a desired time interval for irrigation using the keypad. The ATmega32 microcontroller utilizes Timer0 to count down the specified interval. Once the timer expires, the system checks the soil moisture level using the ADC. If the moisture level is below the threshold, the microcontroller activates the water pump to irrigate the soil.

Benefit: By allowing users to define irrigation intervals, this mode helps in reducing the wear on soil moisture sensors and avoids continuous operation, while still ensuring timely irrigation based on soil moisture conditions.

05
Communication Protocol

Communication Protocol(Data Transmission)

Function: The ATmega32 microcontroller uses its UART peripheral to transmit soil moisture data to a connected laptop or other monitoring device.

Operation: The microcontroller collects soil moisture readings from the sensors and processes this data. Using the UART interface, the microcontroller sends this data in a serial format to the connected device at regular intervals.

Benefit: This enables real-time monitoring of soil moisture levels without needing to manually check the system in the field.

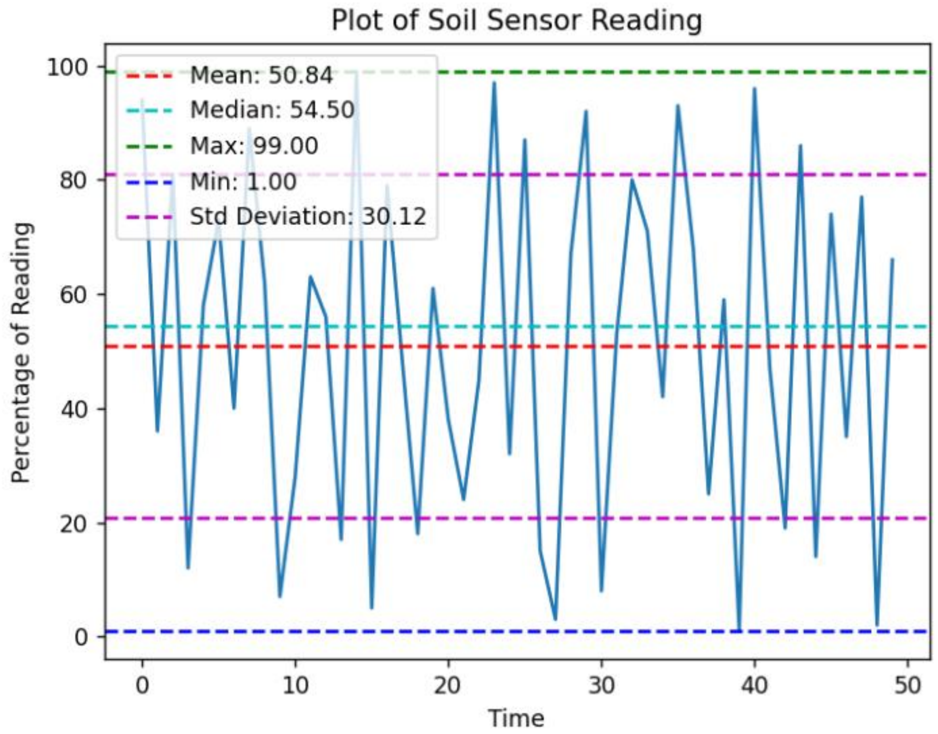
Communication Protocol (Data Capture and Visualization)

Function: Python scripts on the connected laptop are used to capture and visualize the transmitted data.

Operation: A Python script runs on the laptop to continuously receive data from the microcontroller via the UART interface. This script stores the data for logging purposes and processes it to generate real-time charts and graphs.

Benefit: The visualization provides users with comprehensive insights into soil moisture trends over time, helping them make informed decisions about irrigation schedules and crop management.

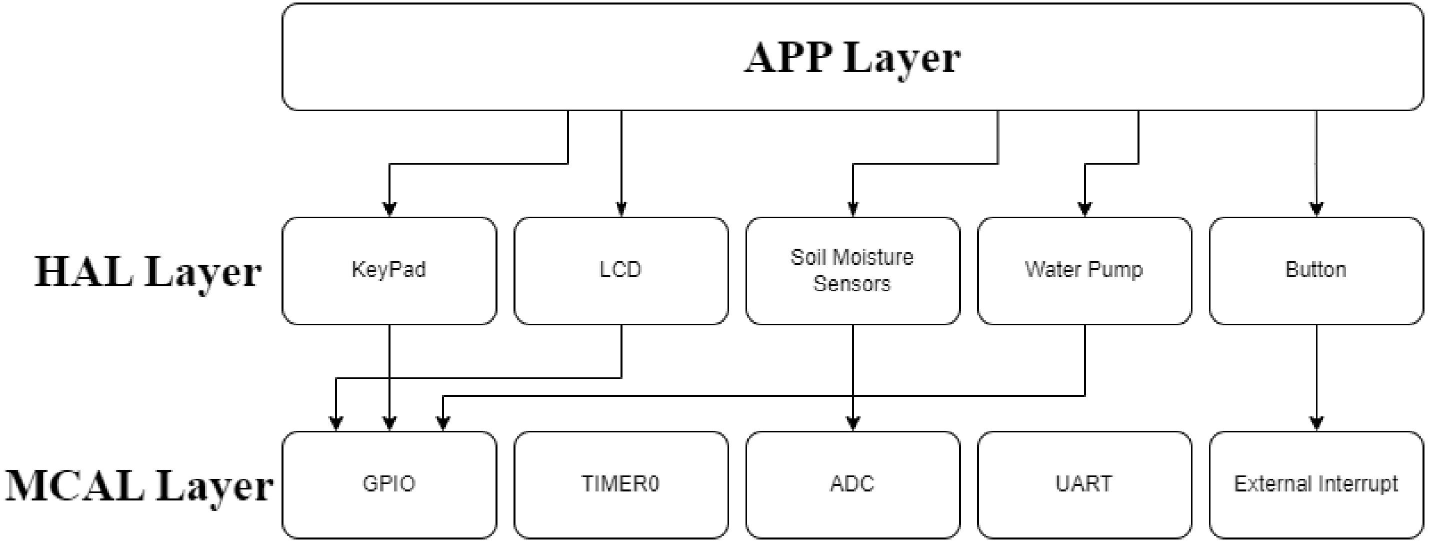
Communication Protocol (Data Capture and Visualization)



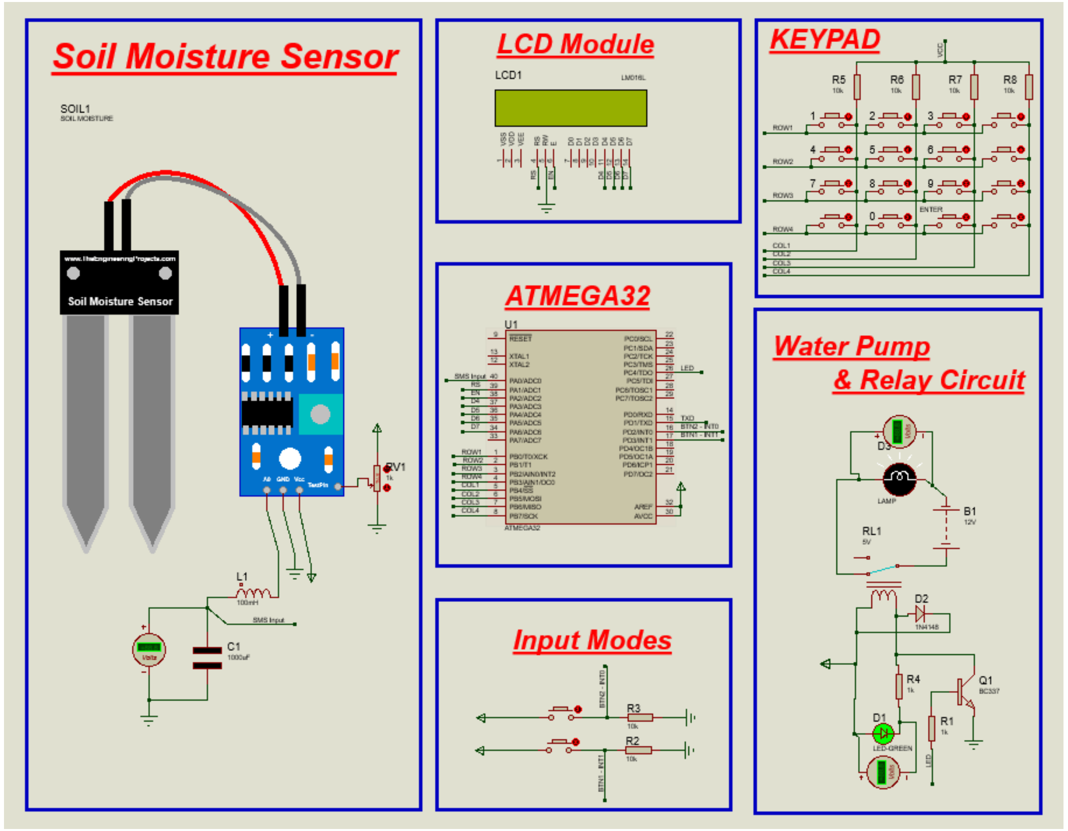
06

Implementation

Design and Planning



Design and Planning(Cont.)



Peripheral Configuration (ADC)

- Using AVCC as the reference voltage, set to 5 volts.
- Using a prescaler of 128 to achieve the required frequency for the ADC in the ATmega32 (less than 125 kHz). Note that the calculation is 16000000/128 = 125 kHz, which allows the ADC to operate within its specified range..

```
/* select the configuration of ADC */
    ADC_ConfigType ADC_Soil =
        {AVCC_5V, ADC_F_CPU_Pre_128};
/* passing the configuration to initialization
    ADC_init(&ADC_Soil);
```

Peripheral Configuration (TIMER0)

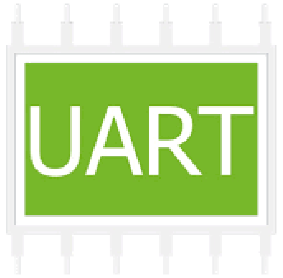
- Using **Compare Match Mode** (CTC) in timer0 and **pre scaler 1024**.
- After using the prescaler, the **timer frequency is 15.625 kHz**, which means **64 μs**.
- Insert the **value 250 into the OCIE0** bit in the **TIMSK** register, and this is the compare match value. When the timer reaches this value, it will generate an interrupt.
- To **count 1 second**, we need **63 interrupts**. The calculation is **250 * 64 μs = 16 ms**, which means for **each interrupt**, the timer **counts 16 milliseconds**. Therefore, for 1 second, we need **1/16 ms = 63 interrupts**.

```
/* select the configuration of TIMER0 */
    TIMER0_ConfigType TIMER0_Config =
        {CTC_INITIAL_VALUE, CTC_VALUE, CTC_MODE, PRESCALER_1024,
         NORMAL_MODE_OC0_DISCONNECTED};
/* passing the configuration to initialization function of TIMER0 */
    Timer0_init(&TIMER0_Config);
```

Peripheral Configuration (UART)

- One stop bit.
- Disable parity bit.
- 8-bit data mode.
- 9600 baud rate.

```
/* select the configuration of UART */
    UART_ConfigType UART_Config =
    { ASYNCHRONOUS, DISABLED_PARITY, STOP_1_BIT, DATA_8_BIT,
      BAUD_RATE_9600 };
/* passing the configuration to initialization function of UART */
    UART_init(&UART_Config);
```

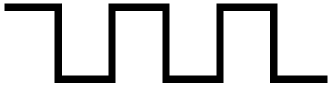


Peripheral Configuration (EXT0 and EXT1)

- Falling edges trigger mode.

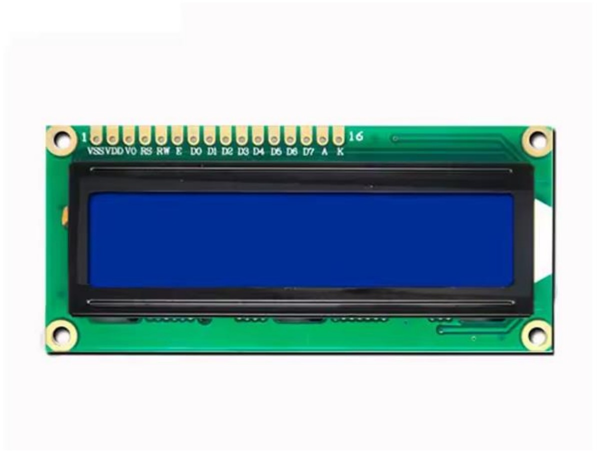
```
/* select the configuration of EXT0 */
    EXT_INT0_ConfigType EXT_INT0_Config =
        {FALLING_EDGE};
/* select the configuration of EXT1 */
    EXT_INT1_ConfigType EXT_INT1_Config =
        {FALLING_EDGE};

/* passing the configuration to initialization function of EXT0 */
    EXT_INT0_init(&EXT_INT0_Config);
/* passing the configuration to initialization function of EXT01 */
    EXT_INT1_init(&EXT_INT1_Config);
```



Peripheral Configuration (GPIO)

- Used to config the LCD, Keypad and Water Pump.



Thanks!

Do you have any questions?

GitHub Repository