Soil Irrigation System (SIS)

Team Members

SN	Student Name	Section Number
1	أحمد حسين بدر رشاد	1
2	أحمد علي فهمي محمد	1
3	أحمد طارق عبد الستار أحمد	1
4	عبد الرحمن هلال صالح عبد العزيز	2
5	عبد الله عيد السيد عمار	2

Embedded Systems Architecture Course, 3rd CSE

Faculty of Engineering, Helwan University

Table of Contents

Table of Figures

1. Preface

1. 1 Document purpose

The purpose of this document is to provide a detailed and complete specification of the soil irrigation system (SIS).

First, there is an overview of the system in the first section the details will appear in the next sections.

1. 2 Target users

- Farmer: primary users who benefit from improved crop yield and reduced manual labor in irrigation.
- Developer: responsible for designing, programming, integrating, and maintaining the system HW and SW components.
- Agricultural Companies: potential collaborators or investors interested in deploying the technology at scale.
- Local Communities: indirectly impacted by improved agricultural practices and water conservation efforts.
- Environmental Agencies: interested in sustainable water management practices.

2. Introduction

2. 1 Purpose

The purpose of the Soil Irrigation System using ATmega32 microcontroller project is to create an efficient and automated solution for optimizing crop growth through precise soil moisture management. By employing a combination of hardware components including the ATmega32 microcontroller, Soil Moisture Sensors, Water Pump, LCD display, Keypad, Relay, and Power Supply, this system aims to accurately monitor soil moisture levels and regulate irrigation accordingly.

The system operates in two distinct modes. In the first mode, the system autonomously senses soil moisture levels using the ADC (Analog-to-Digital Converter) and activates the water pump when the moisture level falls below a predefined threshold (50%). This mode ensures that plants receive adequate water supply without manual intervention.

In the second mode, users can specify a time interval using the keypad, and the system employs Timer0 to count down this period. Upon expiration, the system again utilizes the ADC to assess soil moisture levels. This mode is designed to minimize sensor wear by preventing continuous operation while still ensuring timely irrigation when needed.

Additionally, the project incorporates UART (Universal Asynchronous Receiver-Transmitter) communication to transmit soil moisture data to a laptop. A Python script is utilized to capture this data at regular intervals and another script is employed to visualize the data through chart plotting, providing users with comprehensive insights into soil moisture trends over time.

2. 2 Scope

The scope of the Soil Irrigation System using ATmega32 microcontroller project encompasses a comprehensive range of functionalities and applications aimed at revolutionizing agricultural irrigation practices. The project's scope includes:

- Hardware Development: Designing and assembling the necessary hardware components, including the ATmega32 microcontroller, Soil Moisture Sensors, Water Pump, LCD display, Keypad, Relay, and power supply, to construct a functional soil irrigation system.
- Software Implementation: Programming the ATmega32 microcontroller to manage sensor readings, control the water pump based on soil moisture levels, operate in different modes, and facilitate communication with external devices via UART.

- Mode Selection and Operation: Implementing two distinct modes of operation to cater to different irrigation scenarios. The first mode autonomously monitors soil moisture levels and initiates irrigation, when necessary, while the second mode allows users to set specific time intervals for periodic moisture checks and irrigation.
- Data Transmission and Visualization: Integrating UART communication to transmit real-time soil moisture data to a laptop or external device. Additionally, developing Python scripts to capture and visualize this data through chart plotting, enabling users to analyze soil moisture trends over time.

2. 3 Overview

The Soil Irrigation System project utilizes an ATmega32 microcontroller and various components to automate and optimize crop irrigation. It features two modes: automatic irrigation based on soil moisture levels and user-defined periodic checks. The system communicates data to a laptop via UART for analysis, enabling informed decision-making. This project aims to enhance agricultural efficiency and sustainability while maximizing crop yields.

3. User Requirements Definitions

3. 1 System Functions

1. Soil Moisture Monitoring.
2. Automatic Irrigation Mode.
3. User-Defined Irrigation Mode.
4. Hardware Control.
5. Timer Functionality.
6. Analog-to-Digital Conversion (ADC).
7. UART Communication.
8. External Interrupt Handling.

9. Data Logging.
10. Power Management.

3. 2 Constraints

- Sensor Accuracy: The accuracy and reliability of soil moisture readings from the sensors may be limited, affecting the precision of irrigation decisions.
- Environmental Factors: External environmental factors such as temperature, humidity, and soil composition may impact the system's performance and accuracy.
- User Interface: The usability and effectiveness of the system may be constrained by the simplicity and functionality of the user interface provided by the keypad and LCD display.

4. System Architecture

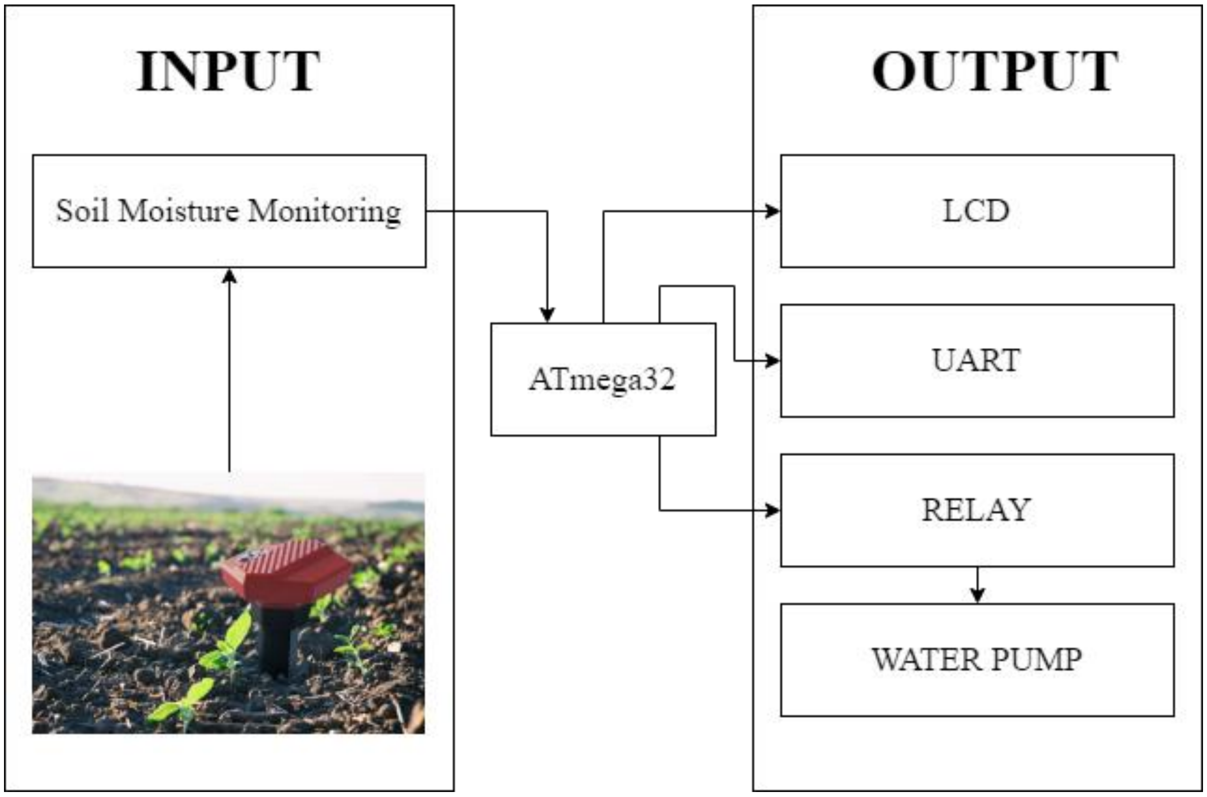

Figure 1: System Architecture

5. System Functional Requirements

5. 1 Soil Moisture Monitoring

The system must continuously monitor soil moisture levels using Soil Moisture Sensors to assess the need for irrigation.

5. 2 Automatic Irrigation Mode

In this mode, the system autonomously activates the Water Pump when soil moisture falls below a predefined threshold (e.g., 50%), ensuring that plants receive adequate water supply without manual intervention.

5. 3 User-Defined Irrigation Mode

This mode allows users to set specific time intervals for periodic soil moisture checks and irrigation. Users can define the frequency of irrigation based on their preferences and specific crop requirements.

5. 4 Hardware Control

The system must effectively control the operation of hardware components such as the Water Pump, LCD display, and Relay using GPIO pins on the microcontroller. Control logic ensures the proper functioning of these components according to the system's requirements.

5. 5 Timer Functionality

Timer functionality, implemented using Timer0, enables the system to manage timing operations, including user-defined irrigation intervals. The timer ensures precise timing for periodic soil moisture checks and irrigation as per user settings.

5. 6 Analog-to-Digital Conversion (ADC)

The system utilizes the microcontroller's ADC module to convert analog soil moisture sensor readings into digital values for processing and analysis. This conversion enables the system to accurately assess soil moisture levels and determine irrigation needs.

5. 7 UART Communication

UART communication allows the system to transmit real-time soil moisture data to a laptop or external device for analysis and visualization. This communication facilitates data exchange between the system and external devices, enabling informed decision-making.

5. 8 External Interrupt Handling

The system utilizes external interrupts to facilitate seamless switching between the two modes of operation. When triggered, the external interrupt prompts the microcontroller to transition from one mode to another based on user input from the Buttons. This functionality enhances user interaction by allowing for convenient mode selection, ensuring flexibility and adaptability in irrigation management.

5. 9 Data Logging

The system logs soil moisture data for analysis and optimization of irrigation schedules. Data logging enables users to track soil moisture trends over time and make informed decisions regarding irrigation management.

5. 10 Power Management

Power management functionality ensures efficient utilization of available power while providing sufficient power for all system components. The system optimizes power consumption to prolong battery life or minimize energy usage, enhancing reliability and sustainability.

6. Peripheral Configuration

6. 1 Analog Digital Converter (ADC)

1. Using AVCC as the reference voltage, set to 5 volts.
2. 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.

6. 2 Timer0

1. Using Compare Match Mode (CTC) in timer0.
2. Using pre scaler 1024.
3. After using the prescaler, the timer frequency is 15.625 kHz, which means 64 microseconds.
4. 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.
5. 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.

6. 3 UART

1. One stop bit.
2. Disable parity bit.
3. 8-bit data mode.
4. 9600 baud rate.

6. 4 EXT0 and EXT1

Faling edges trigger mode.

6. 5 GPIO

Used to config the LCD, Keypad and Water Pump.

7. Pros and Cons

7. 1 Pros

1. Efficiency: The system optimizes water usage by autonomously monitoring soil moisture levels and activating irrigation only when necessary, leading to efficient water usage and reduced water wastage.
2. Automation: Automatic irrigation mode eliminates the need for manual intervention, saving time and effort for farmers while ensuring consistent and timely irrigation for optimal crop growth.
3. Customization: User-defined irrigation mode allows farmers to tailor irrigation schedules to meet the specific needs of different crops, soil types, and environmental conditions, enhancing flexibility and adaptability.
4. Data-driven Decision Making: The system provides real-time soil moisture data for analysis and visualization, enabling informed decision-making regarding irrigation scheduling and resource management.

7. 2 Cons

1. Cost: The initial setup cost of the system, including hardware components and development, may be relatively high, potentially posing a barrier to adoption for small-scale or resource-limited farmers.
2. Complexity: The integration of various hardware components, software functionalities, and communication protocols may introduce complexity in system design, implementation, and maintenance, requiring technical expertise for setup and troubleshooting.
3. Reliability: The reliability of the system may be affected by factors such as sensor accuracy, environmental conditions, and power supply stability, potentially leading to inconsistencies or malfunctions in irrigation operations.

8. Layered Architecture

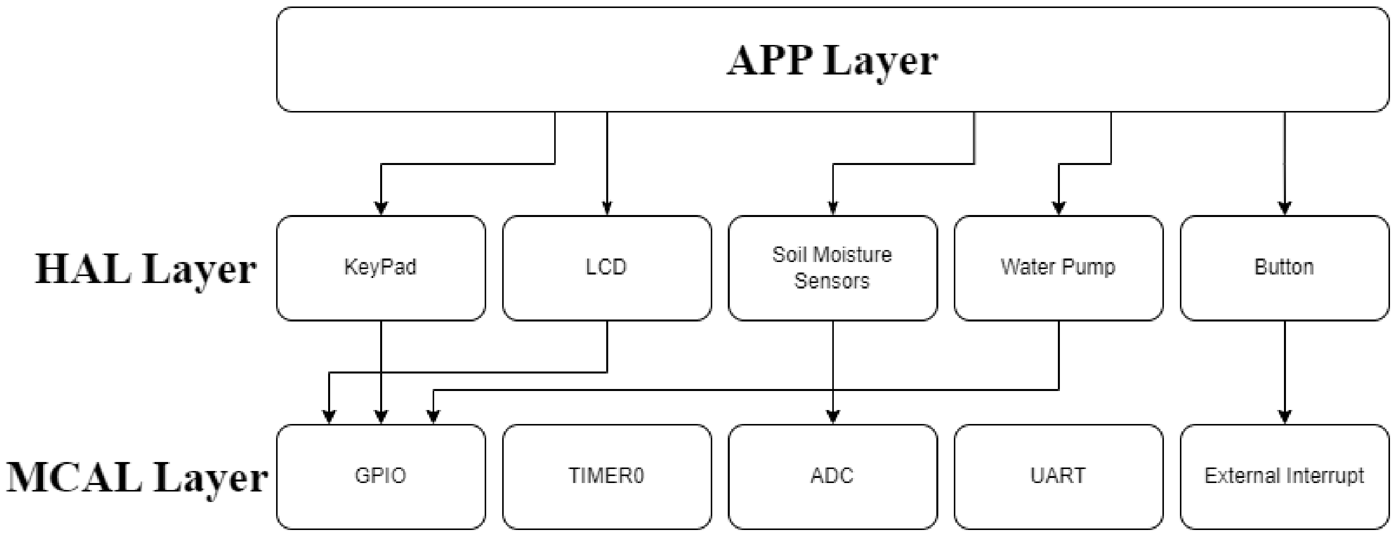

Figure 2: Layered Architecture Diagram.

9. List Of Components

SN	Item Name	Item Type	Item Code Name	Purpose	Quantity
1	ATmega32 Microcontroller	Microcontroller	ATmgea32	Controls system operation based on sensor input and user commands.	1
2	Soil Moisture Sensors	Sensor	SoilMoistureSensor	Measures soil moisture levels for irrigation control	1
3	Water Pump	Actuator	WaterPump	Pumps water for soil irrigation	1
4	LCD Display	Display	LCDDisplay	Provides visual feedback on system status	1
5	Keypad	Input Device	Keypad	Allows user input for system control	1
6	Relay	Switching Device	Relay	Controls high-power components like the water pump	1
7	Power Supply	Power Source	PowerSupply	Provides electrical power to the system	1

8	Tube	Conduit	Tube	Delivers water from the pump to the soil	1
9	Button	Input Device	Button	Used for manual mode switching or additional user input	2

10. Hardware Setup and Data Visualization

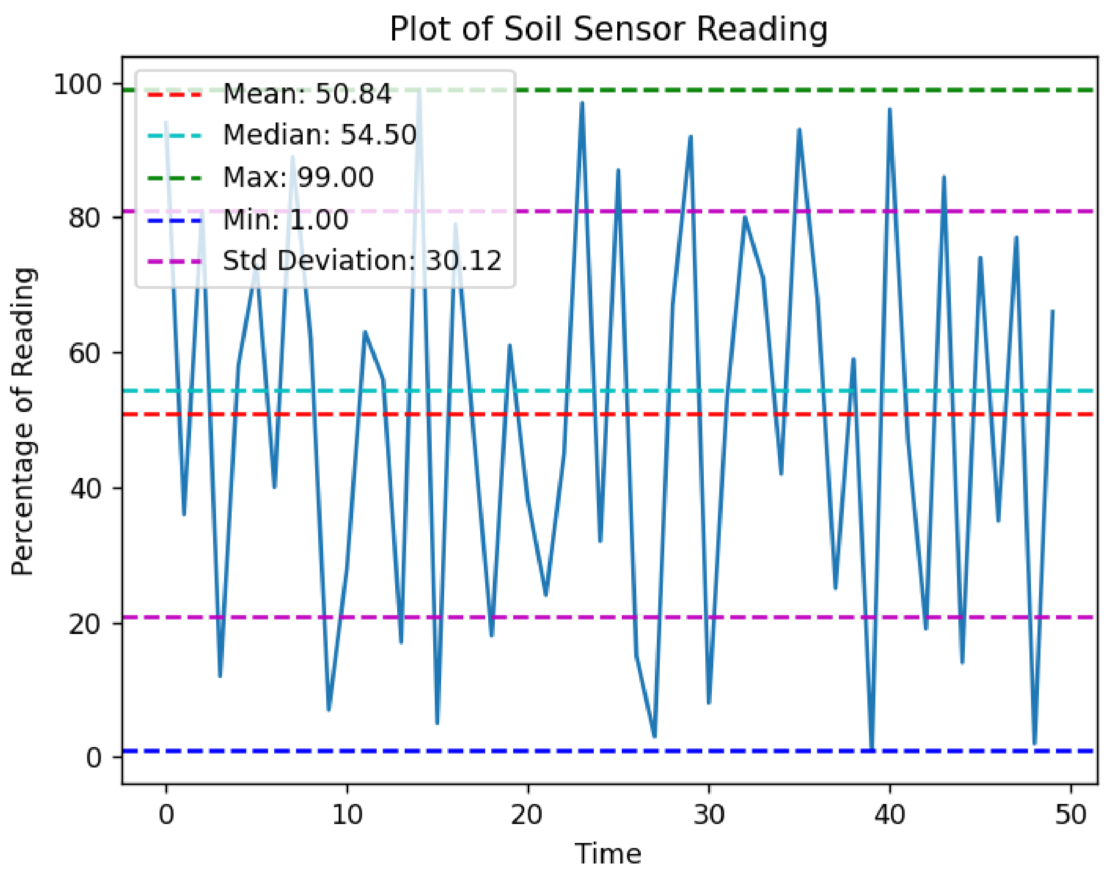

Figure 3: Plot of Sensor Reading

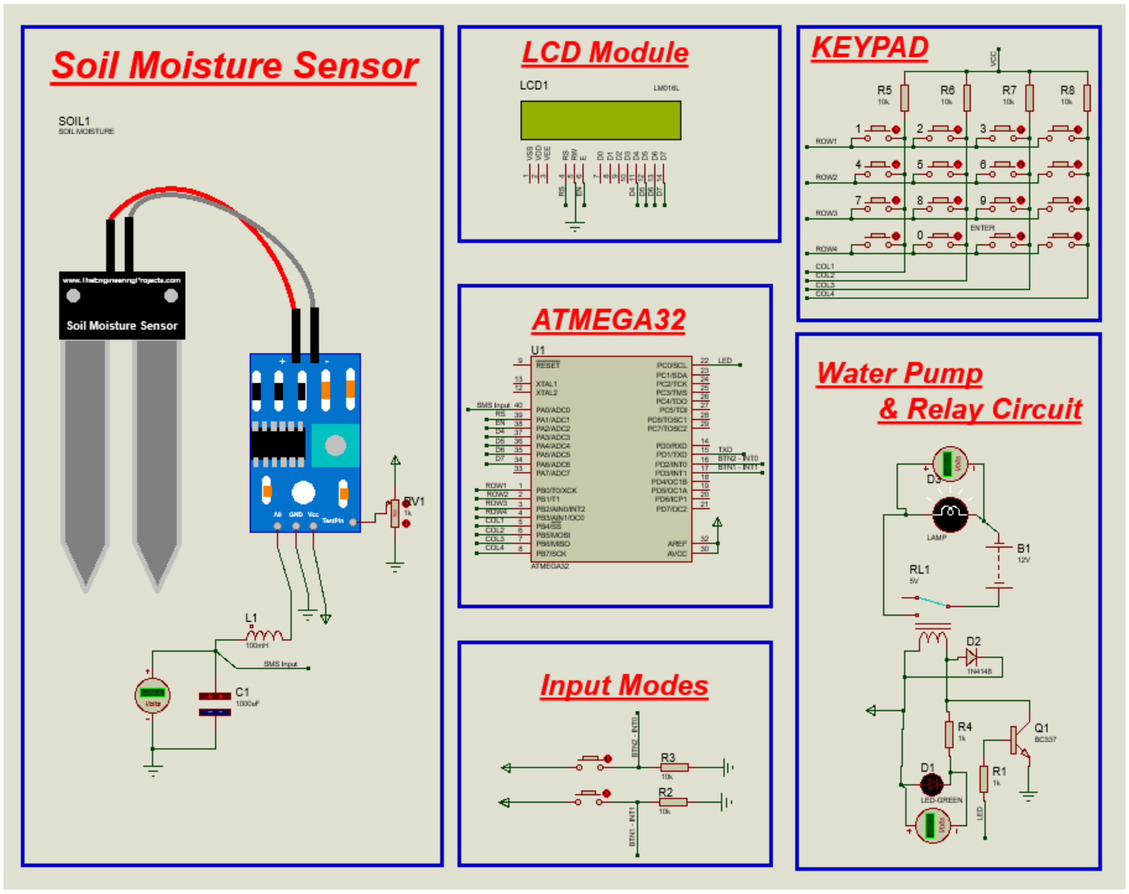

Figure 4: Proteus Simulation

11. Source Code

11. 1 Main File

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

#include "HAL/LCD/lcd.h"
#include "HAL/KEYPAD/keypad.h"
#include "HAL/SOIL_SENSOR/SoilSensor.h"
#include "HAL/WATER_PUMP/waterPumb.h"

#include "MCAL/TIMER0/timer0.h"
#include "MCAL/ADC/adc.h"
#include "MCAL/UART/uart.h"
#include "MCAL/EXTERNAL_INTERRUPT/external_interrupt.h"

#include "LIB/common_macros.h"

/*******************************************************************************
 *                                Definitions                                  *
 *******************************************************************************/

#define PERIOD_LENGTH 2
#define NUMBER_OF_COMPARE_MTACHES_PER_SECOND 63
#define CTC_VALUE                            250
#define CTC_INITIAL_VALUE                    0
#define PERIOD_TO_SEND_DATA_THROUGH_UART     5

/*******************************************************************************
 *                           Global Variables                                  *
 *******************************************************************************/

volatile uint8 g_Mode = 0;
volatile uint8 secondCounter = 0;
volatile uint8 adcValue = 0;
volatile uint8 g_ticks = 0;
volatile uint8 countMinuteFlag = 0;
volatile uint8 counterForMinutes = 5;
volatile uint32 delay_period_copy = 0;
uint32 delay_period = 0;
uint8 percentageOfRead = 0;

/*******************************************************************************
 *                            Functions Prototypes                             *
 *******************************************************************************/
```

```
/*
 * Description :
 * Function to active keypad to get input number from it.
 * Function input:  array of uint8 to save numbers from keypad on it.
 * Function output: void.
 */
void getPeriodFromKeypad(uint8 period[]);

/*
 * Description :
 * This is the call back function which will be executed each interrupt in INT0.
 * Function input:  void.
 * Function output: void.
 */
void INT0_Func(void);

/*
 * Description :
 * This is the call back function which will be executed each interrupt in INT1.
 * Function input:  void.
 * Function output: void.
 */
void INT1_Func(void);

/*
 * Description :
 * This is the call back function which will be executed each interrupt in TIMER0.
 * Function input:  void.
 * Function output: void.
 */
void TIMER0_Func(void);

int main(void)
{
    /* Configuration and initialization functions */

    /* array for the period which user will enter */
    uint8 entredPeriod[PERIOD_LENGTH];

    /* select the configuration of ADC */
    ADC_ConfigType ADC_Soil =
    { AVCC_5V, ADC_F_CPU_Pre_128 };

    /* 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 };

    /* select the configuration of TIMER0 */
    TIMER0_ConfigType TIMER0_Config =
    { CTC_INITIAL_VALUE, CTC_VALUE_FOR_QUARTER_SECOND, CTC_MODE, PRESCALER_1024,
            NORMAL_MODE_OC0_DISCONNECTED };

    /* passing the configuration to initialization function of TIMER0 */
    Timer0_init(&TIMER0_Config);

    /* 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);

    /* passing the configuration to initialization function of ADC */
    ADC_init(&ADC_Soil);

    /* setup the call back function */
    EXT_INT0_setCallBack(INT0_Func);
    /* setup the call back function */
    EXT_INT1_setCallBack(INT1_Func);

    /* setup the call back function */
    Timer0_setCallBack(TIMER0_Func);

    /* initialization function of UART with baud rate 9600 */
    UART_init(9600);

    /* call the initialization function of LCD */
    LCD_init();

    /* call the initialization function of Water Pump */
    WaterPumb_Init();

    /* global interrupt enable */
    SREG |= (1 << 7);

    for (;;)
    {
        /* if the selected mode is the first mode */
        while (g_Mode == 0)
        {
            /* to print the state of water pump and the percentage of ADC reading */
            LCD_displayStringRowColumn(0, 0, "Water Pump: ");
            LCD_displayStringRowColumn(1, 0, "Value: ");
            /* get the value from the ADC channel */
            adcValue = Soilsensor_getValue();
            /* calculate the percentage of reading */
            percentageOfRead = 100 - ((uint32) (adcValue * 100) / 255);
            /* display it in its place in LCD */
            LCD_moveCursor(1, 7);
            LCD_intgerToString(percentageOfRead);
            LCD_displayCharacter('%');
            /*
             * check the state of reading, if less than or equal 50 then activate the
water pump
             * else turn off the water pump
             */
            if (percentageOfRead <= 50)
            {
                LCD_displayStringRowColumn(0, 12, "ON");
                /*
                 * this to only this the character after ON is space
                 * when switch from OFF to ON we need to remove the last F latter
                 */
                LCD_displayCharacter(' ');
                WaterPumb_States(on);
            }
            else
            {
                LCD_displayStringRowColumn(0, 12, "OFF");
                WaterPumb_States(off);
            }
            /* this condition only for LCD display the specific digits from the data  */
            if (percentageOfRead < 100)
            {
                LCD_moveCursor(1, 10);
                LCD_displayCharacter(' ');
            }
            if (percentageOfRead < 10)
            {
                LCD_moveCursor(1, 9);
                LCD_displayCharacter(' ');
            }
        }
        /* if the selected mode is the second mode */
        while (g_Mode == 1)
        {
            /* turn off the pump */
            WaterPumb_States(off);
            LCD_clearScreen();
            LCD_displayStringRowColumn(0, 0, "Enter Period");
            LCD_moveCursor(1, 0);
            /* wait for the user to enter the period */
            getPeriodFromKeypad(entredPeriod);
            _delay_ms(1500);
            LCD_clearScreen();
            LCD_displayStringRowColumn(0, 6, "DONE");
            _delay_ms(1500);
            LCD_clearScreen();
            /* transform delay_period from minutes to seconds */
            delay_period = (entredPeriod[1] + (entredPeriod[0] * 10));
            delay_period *= 60;
            /* make a copy to repeat the operation each delay_period */
            delay_period_copy = delay_period;
            /* continue this mode, its mean we will start count the delay period after
the user enter it */
            g_Mode++;
        }
        while (g_Mode == 2)
        {
            LCD_displayStringRowColumn(0, 0, "Timer : ");
            LCD_intgerToString(delay_period_copy);
            /* this condition only for LCD display the specific digits from the data  */
            if (delay_period_copy < 1000)
            {
                LCD_moveCursor(0, 11);
                LCD_displayCharacter(' ');
            }
            if (delay_period_copy < 100)
            {
                LCD_moveCursor(0, 10);
                LCD_displayCharacter(' ');
            }
            if (delay_period_copy < 10)
            {
                LCD_moveCursor(0, 9);
                LCD_displayCharacter(' ');
            }
            /* check if the counter become 0 which mean the period is done */
            if (delay_period_copy == 0)
            {
                /* check the state of ADC if the soil need water or not */
                adcValue = Soilsensor_getValue();
                percentageOfRead = 100 - ((uint32) (adcValue * 100) / 255);
                /* send the data to UART */
                UART_sendByte(percentageOfRead);
                /* if the percentage less than 50, then the soil need water */
                if (percentageOfRead <= 50)
                {
                    /* active the pump */
                    WaterPumb_States(on);
                    /* make the flag of count one minute is true */
                    countMinuteFlag = 1;
                    /* wait for one minute */
                    while (counterForMinutes != 0)
                        ;
                    /* reset the counter ,flag and turn of the pump */
                    counterForMinutes = 60;
                    countMinuteFlag = 0;
                    WaterPumb_States(off);
                }
                /* repeat the operation each delay_period */
                delay_period_copy = delay_period;
            }
        }
    }
}

/*
 * Description :
 * Function to active keypad to get input number from it.
 * Function input:  array of uint8 to save numbers from keypad on it.
 * Function output: void.
 */
void getPeriodFromKeypad(uint8 period[])
{
    uint8 i, temp; /* i for the period length counter, temp for the number from user */
    i = 0; /* start from index 0 in the array */
    while (i < PERIOD_LENGTH)
    {
        /* take the input number form keypad */
        temp = KEYPAD_getPressedKey();
        /* delay between each press and the next */
        _delay_ms(250);
        /* if the input is not a number form 0 to 9 then repeat taking input step */
        /* ^ for the keys don't have a number or # */
        if (temp == '^' || temp == '#')
        {
            continue;
        }
        /* the input is a number, then save it to its index in array */
        period[i] = temp;
        /* display the the number in LCD */
        LCD_intgerToString(temp);
        /* increment the index */
        i++;
    }
    /* waiting for pressing enter key */
    while (KEYPAD_getPressedKey() != '#')
        ;
}

/*
 * Description :
 * This is the call back function which will be executed each interrupt in INT0.
 * Function input:  void.
 * Function output: void.
 */
void INT0_Func(void)
{
    /* switch the mode for the first mode */
    g_Mode = 0;
}

/*
 * Description :
 * This is the call back function which will be executed each interrupt in INT1.
 * Function input:  void.
 * Function output: void.
 */
void INT1_Func(void)
{
    /* switch the mode for the second mode */
    g_Mode = 1;
}

/*
 * Description :
 * This is the call back function which will be executed each interrupt in TIMER0.
 * Function input:  void.
 * Function output: void.
 */
void TIMER0_Func(void)
{
    g_ticks++;
    /* we need a specific number of ticks for one second delay */
    if (g_ticks == NUMBER_OF_COMPARE_MTACHES_PER_SECOND)
    {
        /* if we in the second mode and not in the water pump activation mode */
        if (g_Mode == 2 && !countMinuteFlag)
        {
            /* then decrement the delay period by 1 */
            delay_period_copy--;
        }
        /* if we in the first mode, we just need to send a data to UART every five
second */
        if (g_Mode == 0)
        {
            secondCounter++;
            if (secondCounter == PERIOD_TO_SEND_DATA_THROUGH_UART)
            {
                /* send the data then reset the second counter */
                UART_sendByte(percentageOfRead);
                secondCounter = 0;
            }
        }
        /* this is the water pump activation mode in mode 2 */
        if (countMinuteFlag)
        {
            counterForMinutes--;
        }
        /* reset the ticks each one second */
        g_ticks = 0;
    }
}
```

11. 2 Python Script to Get Data From UART

```python
import serial
import csv
import struct
import time

# Open serial port
ser = serial.Serial('COM6', baudrate=9600, bytesize=8, parity=serial.PARITY_NONE,
stopbits=1)

# Open CSV file for writing
with open('data.csv', 'a', newline='') as csvfile:
    csvwriter = csv.writer(csvfile)

    print("Listening for data...")

    while True:
        try:
            # Read a line of data from serial port
            data = ser.read().strip()
            if data:
                # Convert byte data to unsigned integer
                value = struct.unpack('B', data)[0]
                print("Received:", value)

                # Write value to CSV file
                csvwriter.writerow([value])
                csvfile.flush()  # Commit changes to the file
                print("Data written to CSV file.")
        except Exception as e:
            print("Error:", e)
```

11. 3 Python Script to Plot Data

```
import csv
import matplotlib.pyplot as plt
import numpy as np

def read_csv(file_path):
    x_values = []
    y_values = []
    with open(file_path, 'r') as file:
        reader = csv.reader(file)
        for i, row in enumerate(reader):
            x_values.append(i)
            y_value = float(row[0])  # Assuming one column in the CSV file
            y_values.append(max(0, min(y_value, 255)))  # Ensure y-value is in the range
[0, 255]
    return x_values, y_values

def plot_data(x_values, y_values):
    plt.plot(x_values, y_values)
    plt.xlabel('Time')
    plt.ylabel('Percentage of Reading')
    plt.title('Plot of Soil Sensor Reading')
    # Calculate statistics
    mean_y = np.mean(y_values)
    median_y = np.median(y_values)
    max_y = np.max(y_values)
    min_y = np.min(y_values)
    std_y = np.std(y_values)
    # Plot statistics
    plt.axhline(y=mean_y, color='r', linestyle='--', label=f'Mean: {mean_y:.2f}')
    plt.axhline(y=median_y, color='c', linestyle='--', label=f'Median: {median_y:.2f}')
    plt.axhline(y=max_y, color='g', linestyle='--', label=f'Max: {max_y:.2f}')
    plt.axhline(y=min_y, color='b', linestyle='--', label=f'Min: {min_y:.2f}')
    plt.axhline(y=mean_y + std_y, color='m', linestyle='--', label=f'Std Deviation:
{std_y:.2f}')
    plt.axhline(y=mean_y - std_y, color='m', linestyle='--')
    # Show legend
    plt.legend()
    plt.show()

if __name__ == "__main__":
    file_path = 'data.csv'  # Replace 'your_csv_file.csv' with the path to your CSV file
    x_values, y_values = read_csv(file_path)
    plot_data(x_values, y_values)
```