KUZEY KIBRIS KAMPÜSÜ
ODTÜ METU
NORTHERN CYPRUS CAMPUS

# CNG 336/EEE 347

# Module 4

# Report


**Name & ID:**

Muhammad Maarij (2486785)

Muhammad Somaan (2528404)


## Declaration

The content of the report represents the work completed by the submitting team only, and no material has been copied from the other source.

## Objectives

The objective of this lab is to:

- Building a complex smart farming system consisting of three MCUs.
- Using interface modules like keypad and LCD.
- Visualizing sensors in proteus environment by employing ADC.
- Applying PWM mode of timers for motor control.

## Introduction

The tasks were divided evenly between the team members. The design, implementation, and verifications were done together.

# Preliminary Questions

**a) The following figure is a block diagram of a successive approximation A/D converter. Explain the purpose of each part of the block diagram shown below. Do you think this A/D converter has parallel or serial digital interface? Explain.**

i. **Sample and Hold:** as the name suggests, it samples (receives) the input signal and holds it constant through the conversion process.
ii. **Comparator:** it compares the input and output signals and accordingly sets the control logic.
iii. **Control Logic:** it provides an approximate digital code to D/A converter.
iv. **D/A Converter:** as the name suggests, it converts digital signal to analog signal. It does so by supplying a reference voltage. Furthermore, it compares it with the input signal.
v. **Shift Register:** shifts the data to right and outputs one bit (LSB) as data out.

Since data out is one bit per cycle, the A/D converter is serial digital interface.

b) **If ATmega128 operates with an MCU clock frequency of 16-MHz, estimate the minimum possible single-ended A/D conversion time, showing corresponding MCU configuration requirements, and calculation. You may ignore the time it takes to initialize the analog circuitry, and may assume free-running mode.**

16 MHz/ 128 = 125 kHz  1/125 kHz = 0.000008s

**c) The following table shows the pre-amplified single-ended voltage range corresponding to the output of each sensor at the remote node, and the corresponding digital values they should be converted to. Since the system uses only 5 bits (based on the protocol described in Lab Module 2) to represent each of the sensed parameters, calculate and fill in the table with the effective resolution of the system in terms of voltage. Also, describe how you may use the existing 10-bit A/D in ATmega128 in obtaining the provided 5-bit adjusted values.**

By reading only the ADCH values in the left adjusted mode, the 10-bit ADC mode was used in 8-bit mode. The relationship to get values according to the provided table was made after defining the relationship between the required digital value and the values read from ADC. Following table shows the calculation done and the answers obtained.

| Parameter | Min. (V) | Max. (V) | Min. (digital) | Max. digital) | Eff. Resolution (mV) |
|---|---|---|---|---|---|
| T | 2 | 4 | 0x03 | 0x1B | $\left(\frac{4-2}{24}=\right)83.33$ |
| M | 1.8 | 4.2 | 0x02 | 0X1E | $\left(\frac{4.2-1.8}{28}=\right)85.71$ |
| W | 2 | 2.8 | 0x04 | 0X1A | $\left(\frac{2.5-2}{22}=\right)22.73$ |
| B | 3 | 5 | 0x01 | 0X01F | $\left(\frac{5-3}{31}=\right)64.52$ |

**d) Given the rotational speed of the water pump (motor) will vary between 20% to 80%, depending on the moisture (M) level at the remote node, calculate and indicate relevant PWM generation settings you plan to program in the motor control section of your remote node solution. What will be the default motor speed before any data has been received from moisture sensor? Why?**

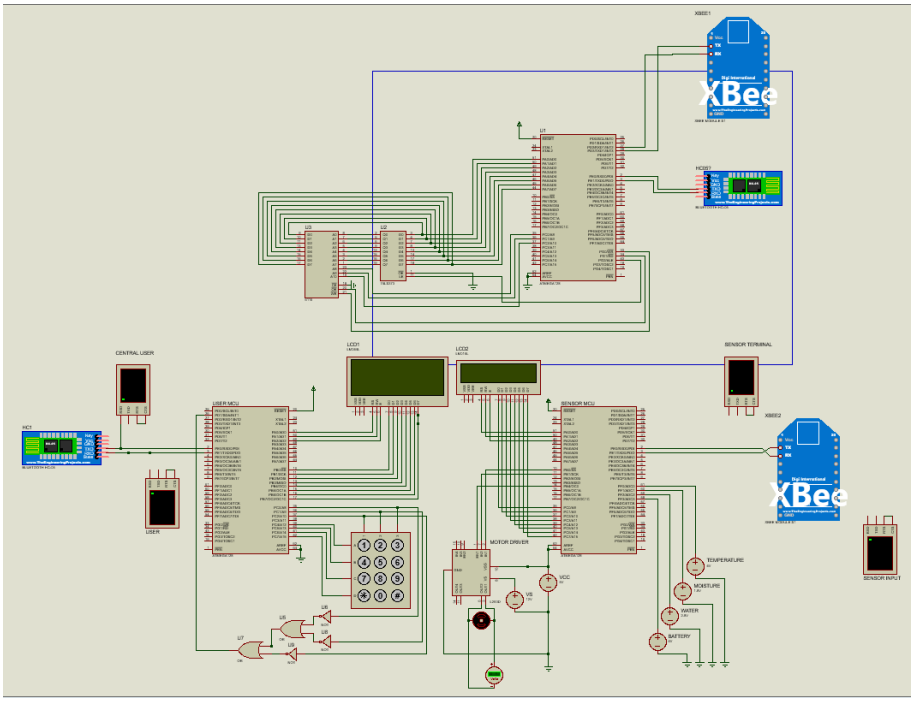
PWM is set by timer0. Phase correct mode and non-inverting mode are used to configure timer0. Phase correct mode is the best choice for controlling the motor as it covers 0%-100% duty cycle range. When no data has been received for moisture level, the OCR0 is set as zero (for zero speed). When moisture level data is read, the OCR0 value is set by mapping.

**e) Outline the main differences between 16x2 LCD discussed in lectures, and 20x4 LCD to be utilized at the user node.**

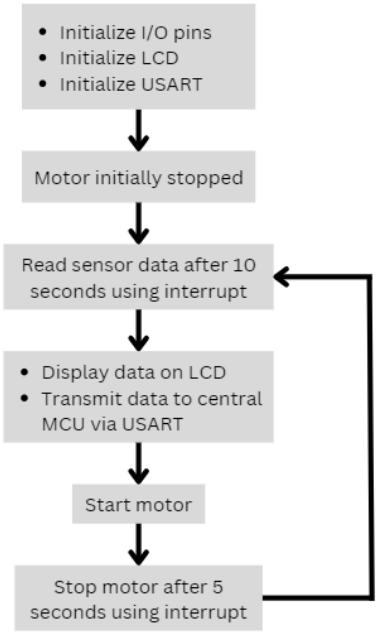
In 16x2 LCD, there are maximum of 32 characters and it has 2 lines. It has 4 and 8 bit modes and commands are same for both with the only difference being the change in line. In 20x4 LCD, there are maximum of 80 characters and it has 4 lines. It has 4 and 8 bit modes and commands are same for both with the only difference being the change in line.

**f) Sketch a system schematic diagram that has the full smart farming system, including 3 ATmega128 MCUs and their connectivity to the peripheral components. Your sketch should be organized and readable, preferably using a drawing application such as Visio. Pin level connectivity should be clear for each pin of each component.**

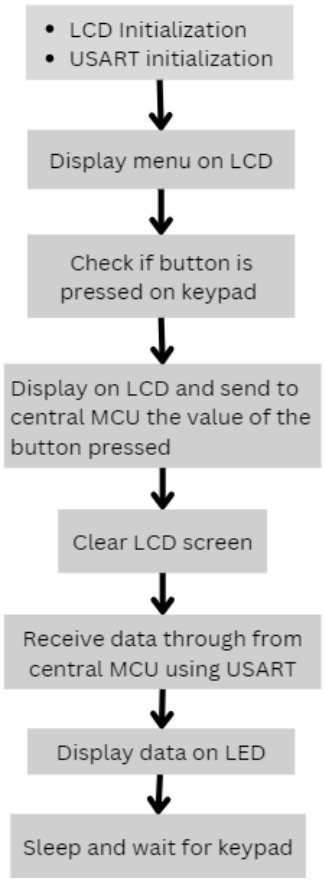
The proteus diagram below shows all connections clearly.



**g) Sketch an algorithmic flowchart to accurately show the program executed in the Remote Sensor Node MCU.**




**h) Sketch an algorithmic flowchart to accurately show the program executed in the User Node MCU.**



**i) Considering your answers to (f-h), and system farming system representation in Figure 4.1, use component datasheets to investigate estimated minimum (IDLE) and maximum (ACTIVE) power dissipation for components in your system, including times when both wireless transmission interfaces are active into your worst-case power scenario. Complete the blanks in Table 4.1.**

| Component Power (mW) | Approx. best-Case (IDLE) | Approx. worst-Case (ACTIVE) |
|---|---|---|
| MCU (Central) | 40 | 85 |
| MCU (User-node) | 40 | 85 |
| MCU (Remote-node) | 40 | 125 |
| Bluetooth Interface | 26.4 | 66 |
| Xbee Interface | 148.5 | 165 |
| 16x2 LCD display | 6 | 6 |
| 20x4 LCD display | 5 | 17.5 |
| Motor driver | 0 | 200 |
| Water pump | 0 | 1000 |

**HC-05 Bluetooth:** for no communication, the voltage is 3.3 V but the current is 8mA for no communication and 20mA while communicating. This gives the best case power dissipation as 26.4mW and worst case as 66mW.

**XBee:** for transmit and receive, the voltage is 3.3 V but the current is 45mA and 50 mA respectively. This gives the best case power dissipation as 148.5mW and worst case as 165mW.

**16x2 LCD:** data from sensors is displayed at all times so the LCD is always active consuming the same amount of power. Assuming 5V and 1.2mA consumption, power is calculated as 6mW.

**20x4 LCD:** the LCD displays something at all times may it be the menu, the request, or the request output. Assuming 5V and 3.5 mA for maximum case and 1 mA for typical case consumption, power is calculated as 5mW for best case and 17.5mW for worst case.

**MCU (central):** the voltage is 5V and the current in the best case (sleep mode) is 8mA and the current in the worst case is 17mA. The power consumption in the best case is 40mW and the worst case is 85mW.

**MCU (user-node):** the voltage is 5V and the current in the best case (sleep mode) is 8mA and the current in the worst case is 17mA. The power consumption in the best case is 40mW and the worst case is 85mW.

**MCU (remote node):** the voltage is 5V and the current in the best case (sleep mode) is 8mA and the current in the worst case is assumed as 25mA. The power consumption in the best case is 40mW and the worst case is 125mW.

# CODE

## Central MCU Code:

```
/*
 * Test 3.c
 *
 * Created: 21-May-23 18:49:07
 * Author : msoma
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <avr/wdt.h>
#include <avr/eeprom.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <util/delay.h>


#define F_CPU 8000000

#define MST_WD_MENU "Enter MS WD Choice:\r1-30ms\r2-250ms\r3-500ms \0"
#define SLV_WD_MENU "Enter SL WD Choice:\r1-500ms\r2-1000ms\r3-2000ms \0"

#define SENSOR_RST_INF_MSG "\rSensor reset ;"

#define USER_MENU "Enter choice:\r1-Mem Dump \r2-Last Entry \r3-Restart \0"

#define USER 0
#define SENSOR 1

#define CRC_POLY 0xD4
#define ONES 0xFF
#define ZEROS 0x00
#define MEM_START 0x500

// Maximum buffer sizes for transmission:
#define USER_TR_BUFFER_SIZE 128
#define SENSOR_TR_BUFFER_SIZE 5
#define STACK_SIZE 100

void MEMORY_INIT();
void BUFFER_INIT();
void USER_BUFFER_INIT();
void SENSOR_BUFFER_INIT();
//Buffer *CREATE_BUFFER(unsigned char size);
void USART_INIT(unsigned char port);
void CONFIG_MASTER_WD();
void ENABLE_MASTER_WD(uint16_t timer);
void CONFIG_SLAVE_WD();
void SLAVE_WD_ENABLE(uint16_t timer);
void SENSOR_INIT();
void APPLY_CRC3(unsigned char input);
unsigned char CRC3(unsigned char input);
unsigned char CRC11(unsigned char input, unsigned char tos);

void HANDLE_MENU_INPUT();
void DUMP_MEMORY();
void LAST_ENTRY();
unsigned char *CONVERT_TO_ASCII(unsigned char toConvert, unsigned char *ascii);

void HANDLE_SENSOR_INPUT(unsigned char input);
void LOG();
unsigned char IS_LOG(unsigned char packet);
unsigned char IS_REPEAT(unsigned char packet);
unsigned char IS_ACKNOWLEDGE(unsigned char packet);
void REPEAT_REQUEST();
```

```
unsigned char CRC11_CHECK(unsigned char input, unsigned char tos);
unsigned char CRC3_CHECK(unsigned char input);
unsigned char IS_COMMAND(unsigned char packet);
unsigned char IS_DATA(unsigned char packet);

void TRANSMIT(unsigned char buffer);
void MESSAGE_USER(const unsigned char *message);
void MESSAGE_SENSOR(const unsigned char *message);
unsigned char TAKE_USER_INPUT();
void ADD_TO_BUFFER(unsigned char buffer, const unsigned char *toAdd);
void SLEEP();

void PUSH(unsigned char val);
unsigned char POP();
unsigned char TOS();

unsigned char userBuffer[USER_TR_BUFFER_SIZE] = "";
unsigned char userBufferSize = 0; // keeps track of character index in buffer
unsigned char userBufferIndex = 0;
unsigned char userBufferCurrent = ' '; // reception one character at a time

unsigned char sensorBuffer[SENSOR_TR_BUFFER_SIZE] = "";
unsigned char sensorBufferSize = 0; // keeps track of character index in buffer
unsigned char sensorBufferIndex = 0;
unsigned char sensorBufferCurrent = ' '; // reception one character at a time

// Data memory pointers
unsigned char *x; // points to the next data memory entry
unsigned char *z; // used in memory dumps


unsigned char *STACK;

unsigned char isStarted = 0;
unsigned char current = 'n';
//Stack* myStack = createStack();

unsigned char crc3Var = 'x';

int main(void)
{
	while (1)
	{
	  /*
	  eeprom_write_byte(0x0000, 0xFF);
	  eeprom_write_byte(0x0001, 0xFF);
	  eeprom_write_byte(0x0002, 0xFF);
	  eeprom_write_byte(0x0003, 0xFF);
	  eeprom_write_byte(0x0004, 0xFF);
	  eeprom_write_byte(0x0005, 0xFF);
	  */
			//unsigned int *x = 0x0500;
			MCUCR = 0x80;
			//tester = eeprom_read_word((const uint16_t *) 0x0000);

			sleep_enable(); // arm sleep mode
			sei(); // global interrupt enable

			MEMORY_INIT();

			BUFFER_INIT();

			USART_INIT(USER);
			USART_INIT(SENSOR);

			CONFIG_MASTER_WD();
			CONFIG_SLAVE_WD();

			SENSOR_INIT();

			MESSAGE_USER(USER_MENU);
```

```
                isStarted = 1;

                while (isStarted)
                {
                        SLEEP();
                }
        }
}

void MEMORY_INIT()
{
        x = (unsigned char *) MEM_START; // points to the next data memory entry
        z = (unsigned char *) MEM_START; // used in memory dumps
        STACK = (unsigned char *) (0x1100 - STACK_SIZE);
}

void BUFFER_INIT()
{
        USER_BUFFER_INIT();
        SENSOR_BUFFER_INIT();
}


void USER_BUFFER_INIT()
{
        userBuffer[USER_TR_BUFFER_SIZE] = "";
        strcpy(userBuffer, "");
        userBufferSize = 0; // keeps track of character index in buffer
        userBufferIndex = 0;
        userBufferCurrent = ' '; // reception one character at a time
}
void SENSOR_BUFFER_INIT()
{
        sensorBuffer[SENSOR_TR_BUFFER_SIZE] = "";
        strcpy(sensorBuffer, "");
        sensorBufferSize = 0; // keeps track of character index in buffer
        sensorBufferIndex = 0;
        sensorBufferCurrent = ' '; // reception one character at a time
}

void USART_INIT(unsigned char port)
{
        if(port == 1)
        {
                //USART1
                //DDRD = 1<<3;
                //PORTD = 0<<2;

                UCSR1B = (1<<TXEN1| 1<<TXCIE1| 1<<RXEN1| 1<<RXCIE1);
                UCSR1C = (1<<UCSZ11)|(1<<UCSZ10);
                UBRR1H = 0x00;
                UBRR1L = 0x33;
        }
        else if(port == 0)
        {
                //USART0
                //DDRE = 1<<3;
                //PORTE.2 = 0b0;

                //Turn on Recieve, Transmission and their interupts
                UCSR0B = (1<<TXEN0 | 1<<TXCIE0 | 1<<RXEN0 | 1<<RXCIE0);
                UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);
                UBRR0H = 0x00;
                UBRR0L = 0x33;
        }
}

void CONFIG_MASTER_WD()
{
        uint16_t timer = eeprom_read_word(0x0000);
```

```
	if(timer == 0xFFFF)
	{
		MESSAGE_USER(MST_WD_MENU);
		unsigned char userInput = ' ';

		while(!(userInput == '1' || userInput == '2' || userInput == '3'))
		userInput = TAKE_USER_INPUT();

		if(userInput == '1')
		timer = 0x001E;
		else if(userInput == '2')
		timer = 0x00FA;
		else if(userInput == '3')
		timer = 0x01F4;

		//unsigned char byte_2 = timer;
		//unsigned char byte_1 = (timer>>8);
		//eeprom_write_byte(0x0000, byte_1);
		//eeprom_write_byte(0x0001, byte_2);
		eeprom_write_word(0x0000, timer);
		eeprom_write_byte(0x0010, 0xFF);

	}
	//ENABLE_MASTER_WD(timer);
}
void ENABLE_MASTER_WD(uint16_t timer)
{
	if(timer == 0x001E)
	wdt_enable(WDTO_30MS);
	else if(timer == 0x00FA)
	wdt_enable(WDTO_250MS);
	else if(timer == 0x01F4)
	wdt_enable(WDTO_500MS);
}

void CONFIG_SLAVE_WD()
{
	uint16_t timer = eeprom_read_word(0x0002);

	if(timer == 0xFFFF)
	{
		MESSAGE_USER(SLV_WD_MENU);
		unsigned char userInput = ' ';

		while(!(userInput == '1' || userInput == '2' || userInput == '3'))
		userInput = TAKE_USER_INPUT();

		if(userInput == '1')
		timer = 0x01F4;
		else if(userInput == '2')
		timer = 0x03E8;
		else if(userInput == '3')
		timer = 0x07D0;

		//unsigned char byte_2 = timer;
		//unsigned char byte_1 = (timer>>8);
		//eeprom_write_byte(0x0002, byte_1);
		//eeprom_write_byte(0x0003, byte_2);
		eeprom_write_word(0x0003, timer);
		eeprom_write_byte(0x0010, 0xFF);

	}
	//SLAVE_WD_ENABLE(timer);
}
void SLAVE_WD_ENABLE(uint16_t timer)
{
	//500 or 1000 or 2000
	//counter value
	TCNT1 = 0;
```

```
	TCCR1A = 0x00;
	TCCR1B = 0b00001100; // CTC with 256 prescalling

	// ms/us = *1000;
	OCR1A = (timer*1000/32);

	TIMSK = (1<<OCIE1A);
}

void SENSOR_INIT()
{
	APPLY_CRC3(0x00);
	PUSH(crc3Var);
	unsigned char pop = POP();
	 _delay_ms(11);
	MESSAGE_SENSOR(&pop);
	//MESSAGE_USER(SENSOR_RST_INF_MSG);
}

void APPLY_CRC3(unsigned char input)
{
	crc3Var = CRC3(input);
	input &= 0b11100000;
	crc3Var |= input;

	//return packet;
}

unsigned char CRC3(unsigned char input)
{
	unsigned char crc = input;
	int counter;
	for(counter = 0; counter<3; counter++)
	{
		if(crc >= 0x80) //most significant bit is 1.
		crc = crc ^ CRC_POLY;

		crc = (crc<<1);
	}
	crc = crc>>3;

	return crc;
}

//unsigned char current_crc = -1;

unsigned char CRC11(unsigned char input, unsigned char tos)
{
	unsigned char crc = tos;
	unsigned char packet = input;
	//current_crc = crc;
	int counter;
	for(counter = 0; counter<11; counter++)
	{
		if(crc >= 0x80)
		crc = crc ^ CRC_POLY;

		crc = (crc<<1);
		unsigned char temp = (0b10000000 & packet);
		packet = packet<<1;
		crc = crc | (temp>>7);
	}
	//TODO MAKE CHANGE
	crc = crc>>3;
	//current_crc = crc;
	return crc;
}

void HANDLE_MENU_INPUT()
{
	unsigned char input = userBuffer[userBufferSize-2];
```

```
		USER_BUFFER_INIT();

		if(input == '1')
			DUMP_MEMORY();
		else if(input == '2')
			LAST_ENTRY();
		else if(input == '3')
			isStarted = 0;

		USER_BUFFER_INIT();
}

//ADD CHECK FOR STACK SKIP IT.
void DUMP_MEMORY()
{
		z = MEM_START;
		while(z < x)
		{
			unsigned char toPrint = *z;
			unsigned char inASCII[2] = "";
			CONVERT_TO_ASCII(toPrint, inASCII);
			MESSAGE_USER(inASCII);
			z++;
		}
}
void LAST_ENTRY() {
		unsigned char toPrint = *(x-1);
		unsigned char inASCII[2] = "";
		CONVERT_TO_ASCII(toPrint, inASCII);
		MESSAGE_USER(inASCII);
		//MESSAGE_USER((x-1));
}

unsigned char *CONVERT_TO_ASCII(unsigned char toConvert, unsigned char *ascii)
{
		//unsigned char ASCII[] = "";
		//unsigned char ascii[3] = "";
		ascii[0] = ((toConvert & 0xF0)>>4);
		ascii[1] = toConvert & 0x0F;


		unsigned char i = 0;
		for(i=0; i<2; i++)
		{
			if(ascii[i]<10)
				ascii[i] |= 0x30;
			else
				ascii[i] = 0x40 | (ascii[i]-9);

			//ASCII[i] = ascii[i];
		}
		ascii[2] = ' ';

		return ascii;
}

void HANDLE_SENSOR_INPUT(unsigned char input)
{
		//unsigned char input = sensorBuffer->current;
		current = input;
		if(IS_COMMAND(input))
		{
			if(IS_DATA(TOS()))
			{
				//DATA IN TOS
				//uint16_t crc = TOS();
				//crc = crc<<8;
				//crc |= input;

				if(CRC11_CHECK(input, TOS()))
```

```
                                {
                                        //PASS
                                        if(IS_LOG(input))
                                        {
                                                LOG();
                                        }
                                }
                                else
                                {
                                        //FAIL
                                        POP();
                                        REPEAT_REQUEST();
                                }
                        }
                        else
                        {
                                //NOT DATA IN TOS
                                if(CRC3_CHECK(input))
                                {
                                        //PASS
                                        if(IS_ACKNOWLEDGE(input))
                                        {
                                                POP();
                                        }
                                        else
                                        {
                                                // NOT ACKNOWLEDGE
                                                if(IS_REPEAT(input))
                                                {
                                                        if(TOS() != 0x00)
                                                        {
                                                                unsigned char tos = TOS();
                                                                MESSAGE_SENSOR(&tos);
                                                        }
                                                }
                                        }
                                }
                                else
                                {
                                        //FAIL
                                        REPEAT_REQUEST();
                                }
                        }
                }
                else
                {
                        //NO COMMAND
                        //ADD TO TOS
                        PUSH(input);
                }
                //NO

                //SENSOR_BUFFER_INIT();
}

void LOG()
{
        *(x++) = TOS();
        POP();
        PUSH(0x40);
        APPLY_CRC3(TOS());
        MESSAGE_SENSOR(&crc3Var);
}

unsigned char IS_LOG(unsigned char packet) {
        packet &= 0b11100000;
        if(packet == 0x20)
                return 1;
        else
                return 0;
}
```

```
unsigned char IS_REPEAT(unsigned char packet) {
	packet &= 0b11100000;
	if(packet == 0x60)
		return 1;
	else
		return 0;
}
unsigned char IS_ACKNOWLEDGE(unsigned char packet) {
	packet &= 0b11100000;
	if(packet == 0x40)
		return 1;
	else
		return 0;
}
unsigned char IS_COMMAND(unsigned char packet) {
	return (packet < 0x80);
}
unsigned char IS_DATA(unsigned char packet) {
	return (packet >= 0x80);
}

void REPEAT_REQUEST() {
	APPLY_CRC3(0x60);
	MESSAGE_SENSOR(&crc3Var);
}

unsigned char CRC11_CHECK(unsigned char input, unsigned char tos) {
	return (CRC11(input, tos) == 0x00);
}
unsigned char CRC3_CHECK(unsigned char input) {
	return (CRC3(input) == 0x00);
}



void TRANSMIT(unsigned char buffer)
{
	if(buffer == USER)
	{
		while(userBufferIndex < userBufferSize)
		{
			UDR0 = userBuffer[userBufferIndex++];
			SLEEP();
			//_delay_ms(1000);
			//while((UCSR0A & (1<<UDRE0)) != 1);
		}
	}
	else
	{
		while(sensorBufferIndex < sensorBufferSize)
		{
			UDR1 = sensorBuffer[sensorBufferIndex++];
			SLEEP();
		}
	}

}
void MESSAGE_USER(const unsigned char *message)
{
	ADD_TO_BUFFER(USER, message);
	TRANSMIT(USER);
}

void MESSAGE_SENSOR(const unsigned char *message)
{
	ADD_TO_BUFFER(SENSOR, message);
	TRANSMIT(SENSOR);
}

unsigned char TAKE_USER_INPUT()
{
```

```
	while(userBufferCurrent != '.')
	SLEEP();

	unsigned char userInput = userBuffer[userBufferSize-2];

	USER_BUFFER_INIT();
	return userInput;
}
void ADD_TO_BUFFER(unsigned char buffer, const unsigned char *toAdd)
{
	if(buffer == USER)
	{
		strcpy(userBuffer, strcat(userBuffer, toAdd));
		//strcpy(userBuffer, toAdd);
		userBufferSize = strlen(userBuffer);
	}
	else
	{
		strcpy(sensorBuffer, strcat(sensorBuffer, toAdd));
		sensorBufferSize = strlen(sensorBuffer);
		if(toAdd[0] == 0x00)
			sensorBufferSize++;
	}

}

void SLEEP()
{
	sleep_enable(); // arm sleep mode
	sei(); // global interrupt enable
	sleep_cpu(); // put CPU to sleep
}



ISR(TIMER1_COMPA_vect) //iv IVT_ADDR_TIMER1_COMPA //ISR
{
	SENSOR_INIT();
}
// Sample Interrupt handler for USART0 (User side) when a TX is done
ISR(USART0_TX_vect)
{
	sleep_disable();

	//CHANGE TO 0 if going --;
	if (userBufferIndex == userBufferSize)
	{ // transmit buffer is emptied
		USER_BUFFER_INIT(); // reinitialize the buffer for next time
		//UCSR0B &= ~((1 << TXEN0) | (1 << TXCIE0)); // disableinterrupt
	}
}

// Sample Interrupt handler for USART0 (User side) when an RX is done
ISR(USART0_RX_vect)
{
	sleep_disable();

	while (! (UCSR0A & (1<<RXC0))){}; // Double checking    flag

	userBufferCurrent = UDR0;
	ADD_TO_BUFFER(USER, &userBufferCurrent);

	if(userBufferCurrent == '.' && isStarted)
	HANDLE_MENU_INPUT();

}


//SENSOR ISR
ISR(USART1_TX_vect)
{
```

```
	sleep_disable();

	if (sensorBufferIndex == sensorBufferSize)
		SENSOR_BUFFER_INIT();
}
ISR(USART1_RX_vect)
{
	sleep_disable();

	while (! (UCSR1A & (1<<RXC1))){}; // Double checking    flag

	//sensorBuffer->current = UDR0;
	//ADD_TO._BUFFER(sensorBuffer, UDR0);

	HANDLE_SENSOR_INPUT(UDR1);
}



void PUSH(unsigned char val)
{
	if(STACK== (unsigned char *) 0x10CF)
		STACK--;
	*(STACK++) = val;
}
unsigned char POP() {
	if(STACK >= (unsigned char *) 0x1100-STACK_SIZE)
		return *(--STACK);
	return 0x00;
}
unsigned char TOS() {
	if(STACK >= (unsigned char *) 0x1100-STACK_SIZE)
		return *(STACK-1);
	return 0x00;
}
```

## Sensor Code:

```
/*
 * UserNode.c
 *
 * Created: 26-May-23 20:12:03
 *  Author: msoma
 */
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <avr/wdt.h>
#include <avr/eeprom.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <util/delay.h>


#define F_CPU 8000000

#define LCD_2_LINE_MODE 0x38
#define DISPLAY_ON 0x08
#define CLEAR_DISPLAY 0x01
#define SHIFT_AFTER_DISPLAY 0x06
#define DISPLAY_BLINK 0x0C
#define DISPLAY_ON_BLINK 0x0F


#define TEMP_MAX 0x1B
#define TEMP_MIN 0x03
#define TEMP_MAX_VOLT 4.0
#define TEMP_MIN_VOLT 2.0
```

```
#define TEMP_VAL_INDEX 4

#define MOIST_MAX 0x1E
#define MOIST_MIN 0x02
#define MOIST_MAX_VOLT 4.2
#define MOIST_MIN_VOLT 1.8
#define MOIST_VAL_INDEX 14

#define WATER_MAX 0x1A
#define WATER_MIN 0x04
#define WATER_MAX_VOLT 2.8
#define WATER_MIN_VOLT 2.0
#define WATER_VAL_INDEX 20

#define BAT_MAX 0x1F
#define BAT_MIN 0x01
#define BAT_MAX_VOLT 5.0
#define BAT_MIN_VOLT 3.0
#define BAT_VAL_INDEX 30

#define BAT_LOW 3.2
#define VCC 5.0

#define MOTOR_MAX_SPEED 80
#define MOTOR_MIN_SPEED 20

#define CRC_POLY 0xD4

//HERE
void START_INIT();
void PORTS_INIT();
void USART_INIT();
void LCD_INIT();
void VARS_INIT();

void sendCommand(unsigned char command);
void sendData(unsigned char data);
void sendLCD(unsigned char isCommand, unsigned input);

void TRANSMIT(unsigned char message);

void CONFIG_SENSOR_TIMER();
void CONFIG_MOTOR_TIMER();
//void RESET_MOTOR_TIMER();
void RESET_SENSOR_TIMER();

void UPDATE_DISPLAY_MESSAGE();
unsigned char *CONVERT_TO_ASCII(unsigned char toConvert, unsigned char *ascii);


void SLEEP();

void START_SENSOR_CONVERSION();
void TEMP_ADC_SENSOR();
void MOIST_ADC_SENSOR();
void WATER_ADC_SENSOR();
void BAT_ADC_SENSOR();
void CONVERSION_COMPLETE();
void SEND_NEXT();
unsigned char CALCULATE_MOTOR_SPEED();
float ADC_VALUE(float volt);


void APPLY_CRC3(unsigned char input);
unsigned char CRC3(unsigned char input);
unsigned char CRC3_CHECK(unsigned char input);
unsigned char CRC11(unsigned char input, unsigned char tos);
unsigned char APPLY_CRC11(unsigned char data, unsigned char command);

void TRANSMIT(unsigned char buffer);
```

```
unsigned char isBatLow = 0;

unsigned char totalChars = 0;

unsigned char isStarted = 0;
unsigned char dataSent = 0x00;
unsigned char sensorInteruptCount = 0;


unsigned char displayMessage[32] = "T=0xVV    M=0xVVW=0xVV    B=0xVV";
unsigned char displayBatteryLow[32] = " Change Battery | Immediately | ";
unsigned char sensorData[4] = "";
unsigned char currentSensorConversion = 0;

unsigned char minValue[4] = "";
unsigned char maxValue[4] = "";
float maxVoltsValue[4];
float minVoltsValue[4];
float resolution[4];
float batteryLowVal;

unsigned char crc3Var = 'x';
unsigned char acknowledge;
unsigned char currentSentIndex = 0;
unsigned char reset = 0;

int main(void)
{
                        START_INIT();


                        //Enable INT1.
                        //GICR = 0x80;
                        //Enable ISC11 and ISC10. Rising edge on INT 1.
                        //EICRA = 0x0C;
                        //EIMSK = 0x02;

                        //sendData('R');
                        //sendData('E');
                        //sendData('S');
                        //lastChar = '\0';
                        //sendData('\0');


                        isStarted = 1;
                        //sei();
                        while (isStarted)
                        {
                                SLEEP();
                                //CHECK_INPUT();
                        }

}

void START_INIT()
{
        PORTS_INIT();
        LCD_INIT();
        USART_INIT();
        VARS_INIT();
        //RESET_MOTOR_TIMER();
        //RESET_SENSOR_TIMER();
}

void VARS_INIT()
{
        isBatLow = 0;
        totalChars = 0;
        isStarted = 0;
```

```
	dataSent = 0x00;
	sensorInteruptCount = 0;

	//displayMessage[32] = "T=0xVV    M=0xVVW=0xVV    B=0xVV";
	//displayBatteryLow[32] = " Change Battery | Immediately | ";
	strcpy(sensorData, "    ");
	currentSensorConversion = 0;


	maxVoltsValue[0] = ADC_VALUE(TEMP_MAX_VOLT);
	minVoltsValue[0] = ADC_VALUE(TEMP_MIN_VOLT);

	maxVoltsValue[1] = ADC_VALUE(MOIST_MAX_VOLT);
	minVoltsValue[1] = ADC_VALUE(MOIST_MIN_VOLT);

	maxVoltsValue[2] = ADC_VALUE(WATER_MAX_VOLT);
	minVoltsValue[2] = ADC_VALUE(WATER_MIN_VOLT);

	maxVoltsValue[3] = ADC_VALUE(BAT_MAX_VOLT);
	minVoltsValue[3] = ADC_VALUE(BAT_MIN_VOLT);

	resolution[0] = ((maxVoltsValue[0]-minVoltsValue[0])/(TEMP_MAX-TEMP_MIN));
	resolution[1] = ((maxVoltsValue[1]-minVoltsValue[1])/(MOIST_MAX-MOIST_MIN));
	resolution[2] = ((maxVoltsValue[2]-minVoltsValue[2])/(WATER_MAX-WATER_MIN));
	resolution[3] = ((maxVoltsValue[3]-minVoltsValue[3])/(BAT_MAX-BAT_MIN));

	minValue[0] = TEMP_MIN;
	minValue[1] = MOIST_MIN;
	minValue[2] = WATER_MIN;
	minValue[3] = BAT_MIN;

	maxValue[0] = TEMP_MAX;
	maxValue[1] = MOIST_MAX;
	maxValue[2] = WATER_MAX;
	maxValue[3] = BAT_MAX;

	batteryLowVal = ADC_VALUE(BAT_LOW);

	if(!reset)
		acknowledge = 0;
	currentSentIndex = 0;
	reset = 0;
}


void PORTS_INIT()
{
	/*
	DDRA.0 = RS
	DDRA.1 = RW
	DDRA.2 = E
	*/
	DDRA = 0xFF;

	//Motor output
	DDRB = 0xFF;
	PORTB = 1;

	//LCD DATA OUTPUT
	DDRC = 0xFF;

	//SENSOR Input.
	DDRF = 0x00;
}

void USART_INIT()
{
	UCSR0B = (1<<TXEN0 | 1<<TXCIE0 | 1<<RXEN0 | 1<<RXCIE0);
	UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);
	UBRR0H = 0x00;
	UBRR0L = 0x33;
```

```
	sei();
}
void LCD_INIT()
{
	totalChars = 0;
	//2 Line display:
	//https://morf.lv/simple-library-for-driving-20x4-lcd-with-4bits
	sendCommand(LCD_2_LINE_MODE);
	sendCommand(DISPLAY_ON_BLINK);
	sendCommand(CLEAR_DISPLAY);
	sendCommand(SHIFT_AFTER_DISPLAY);
	_delay_ms(10);
}




void TRANSMIT(unsigned char message)
{
	dataSent = message;
	UDR0 = message;
}


void sendCommand(unsigned char command) {
	sendLCD(0, command);
}

void sendData(unsigned char data)
{
	sendLCD(1, data);
	totalChars++;

	unsigned char lineNum = totalChars/16;
	unsigned char currentChars = totalChars%16;
	unsigned char remainingChars = 16-currentChars;

	if(data == '\r' && currentChars != 0)
	{
		totalChars += remainingChars;
		lineNum++;
	}

	if(remainingChars==16 || data == '\r')
	{
		if(lineNum == 1)
			sendCommand(0xC0);
		//else if(lineNum == 2)
			//LCD_INIT();
	}
}

void sendLCD(unsigned char isData, unsigned input)
{
	PORTA = 0x00+isData;
	PORTC = input;

	PORTA = 0x04+isData;
	_delay_ms(1);
	PORTA = 0x00+isData;
	_delay_ms(1);
}



ISR(USART0_TX_vect) {
	sleep_disable();
}

// Sample Interrupt handler for USART0 (User side) when an RX is done
```

```
ISR(USART0_RX_vect)
{
	sleep_disable();

	while (! (UCSR0A & (1<<RXC0))){}; // Double checking    flag

	unsigned char commandIn = UDR0;

	if(!CRC3_CHECK(commandIn))
	{
		APPLY_CRC3(0x60);
		TRANSMIT(crc3Var);
		return;
	}

	commandIn &= 0b11100000;

	if(commandIn == 0x00) //RESET
	{
		//START_INIT();
		CONFIG_SENSOR_TIMER();
		CONFIG_MOTOR_TIMER();
		APPLY_CRC3(0x40);
		TRANSMIT(crc3Var);
		reset = 1;
		acknowledge = 1;
	}
	else if(commandIn == 0x40) //Acknowledge
	{
		acknowledge = 1;
		SEND_NEXT();
	}
	else if(commandIn == 0x60)
	{
		acknowledge = 1;
		currentSensorConversion--;
		SEND_NEXT();
	}
	//sendData(UDR0);
}


void SLEEP()
{
	sleep_enable(); // arm sleep mode
	sei(); // global interrupt enable
	sleep_cpu(); // put CPU to sleep
}







void CONFIG_SENSOR_TIMER()
{
	//500 or 1000 or 2000
	//counter value
	TCNT1 = 0;
	TCCR1A = 0x00;
	TCCR1B = 0b00001101; // CTC with 1024 prescalling

	// (1/8Mhz)*1024 = 128us

	// s/us = *1000000;
	OCR1A = (5*1000000/128);

	TIMSK = (1<<OCIE1A);
}
```

```
ISR(TIMER1_COMPA_vect) //iv IVT_ADDR_TIMER1_COMPA //ISR
{
	sleep_disable();

	sensorInteruptCount++;
	if(sensorInteruptCount%2)
	{
		//TURN OFF MOTOR.
		OCR0 = 0;
		return;
	}

	START_SENSOR_CONVERSION();
}

void UPDATE_DISPLAY_MESSAGE()
{
	unsigned char toASCII[3] = "";
	CONVERT_TO_ASCII(sensorData[0], toASCII);
	displayMessage[TEMP_VAL_INDEX] = toASCII[0];
	displayMessage[TEMP_VAL_INDEX+1] = toASCII[1];

	CONVERT_TO_ASCII(sensorData[1], toASCII);
	displayMessage[MOIST_VAL_INDEX] = toASCII[0];
	displayMessage[MOIST_VAL_INDEX+1] = toASCII[1];

	CONVERT_TO_ASCII(sensorData[2], toASCII);
	displayMessage[WATER_VAL_INDEX] = toASCII[0];
	displayMessage[WATER_VAL_INDEX+1] = toASCII[1];

	CONVERT_TO_ASCII(sensorData[3], toASCII);
	displayMessage[BAT_VAL_INDEX] = toASCII[0];
	displayMessage[BAT_VAL_INDEX+1] = toASCII[1];
}

void CONFIG_MOTOR_TIMER()
{
	//500 or 1000 or 2000
	//counter value
	TCNT0 = 0;
	TCCR0 = 0b01100001;		//PWM Phase correct, non-inverting.
	//TCCR3B = 0b00001101; // CTC with 1024 prescalling

	// (1/8Mhz)*1024 = 128us

	// 5 seconds.
	// s/us = *1000000;
	OCR0 = 0;

	//TIMSK |= (1<<OCIE3A);
}

/*
void RESET_MOTOR_TIMER()
{
	TCNT3 = 0;
	TCCR3A = 0x00;
	TCCR3B = 0x00;
	OCR3A = 0x00;
	TIMSK &= ~(1<<OCIE3A);
}
*/
void RESET_SENSOR_TIMER()
{
	TCNT1 = 0;
	TCCR1A = 0x00;
	TCCR1B = 0x00;
	OCR1A = 0x00;
	TIMSK &= ~(1<<OCIE1A);
}
```

```
/*
ISR(TIMER0_COMPA_vect) //iv IVT_ADDR_TIMER1_COMPA //ISR
{
	sleep_disable();
	//RESET_MOTOR_TIMER();

	//TODO: DISABLE MOTOR.
}
*/

unsigned char *CONVERT_TO_ASCII(unsigned char toConvert, unsigned char *ascii)
{
	//unsigned char ASCII[] = "";
	//unsigned char ascii[3] = "";
	ascii[0] = ((toConvert & 0xF0)>>4);
	ascii[1] = toConvert & 0x0F;


	unsigned char i = 0;
	for(i=0; i<2; i++)
	{
		if(ascii[i]<10)
		ascii[i] |= 0x30;
		else
		ascii[i] = 0x40 | (ascii[i]-9);

		//ASCII[i] = ascii[i];
	}
	ascii[2] = ' ';

	return ascii;
}

void START_SENSOR_CONVERSION()
{
	//Enable ADC and clk 128. and interupt.
	//ADCSRA = 0x87;
	ADCSRA = 0x8F;
	//Right adjusted and internal 2.56 ref.
	//ADMUX = 0b11000000;


	//ADCSRA |= (1<<ADSC); // start conversion

	//while ((ADCSRA & (1<<ADIF))==0);
	// wait for end of conversion
	//ADCSRA |= (1<<ADIF);

	//sensorData[0] = ADCH;

	//CONVERSION_COMPLETE();
	TEMP_ADC_SENSOR();
}

ISR(ADC_vect)
{
	sleep_disable();

	unsigned char currentADC = ADCH;
	float result = ((currentADC-
minVoltsValue[currentSensorConversion])/resolution[currentSensorConversion])+minValue[currentSensorConversion]+1;

	sensorData[currentSensorConversion] = result;

	if(ADCH<minVoltsValue[currentSensorConversion])
		sensorData[currentSensorConversion] = minValue[currentSensorConversion];
	else if(ADCH>maxVoltsValue[currentSensorConversion])
		sensorData[currentSensorConversion] = maxValue[currentSensorConversion];

	if(currentSensorConversion==3)
```

```
                        if(currentADC<=batteryLowVal)
                                isBatLow = 1;

        currentSensorConversion++;

        ADC = 0x0000;
        ADCSRA |= (1<<ADIF);
        ADCSRA &= ~(1<<ADSC);
        //currentSensorConversion++;

        if(currentSensorConversion==1)
                MOIST_ADC_SENSOR();
        else if(currentSensorConversion==2)
                WATER_ADC_SENSOR();
        else if(currentSensorConversion==3)
                BAT_ADC_SENSOR();
        else
                CONVERSION_COMPLETE();

}

void SEND_NEXT()
{
        //No acknowledge recieved or
        if(currentSentIndex >= 4 || acknowledge == 0)
                return;

        unsigned char dataPacket = sensorData[currentSentIndex];
        dataPacket |= currentSentIndex<<5;
        dataPacket |= 1<<7;

        printf("%d 's datapacket = %d\n", currentSentIndex, dataPacket);

        unsigned char command = APPLY_CRC11(dataPacket, 0b00100000);
        //TODO COMMAND
        TRANSMIT(dataPacket);
        _delay_ms(1);
        TRANSMIT(command);

        currentSentIndex++;
        acknowledge = 0;
}

void CONVERSION_COMPLETE()
{
        ADCSRA = 0x00;
        currentSensorConversion = 0;
        currentSentIndex = 0;

        //CONFIG_MOTOR_TIMER();

        //TODO
        //START MOTOR..
        unsigned char motorSpeed = CALCULATE_MOTOR_SPEED();
        OCR0 = motorSpeed;


        UPDATE_DISPLAY_MESSAGE();

        unsigned char i = 0;

        LCD_INIT();

        for(i=0; i<32; i++)
        {
                if(isBatLow)
                        sendData(displayBatteryLow[i]);
                else
                        sendData(displayMessage[i]);
        }
```

```
		//TRANSMIT DATA.
		SEND_NEXT();
}


//TODO ALL THE SENSORS
void TEMP_ADC_SENSOR()
{
		// 2.56 V Vref, ADC0 single ended  data will be left-justified
		ADMUX = 0x60;
		//ADMUX |= (1<<MUX0);
		//Start Conversion
		ADCSRA |= (1<<ADSC);

		//return 0x00;
}
void MOIST_ADC_SENSOR()
{
		ADMUX = 0x60;
		//ADC 1 selected.
		ADMUX |= (1<<MUX0);
		//Start Conversion
		ADCSRA |= (1<<ADSC);

		//return 0x00;
}
void WATER_ADC_SENSOR()
{
		ADMUX = 0x60;
		//ADC 2 selected.
		ADMUX |= (1<<MUX1);
		//Start Conversion
		ADCSRA |= (1<<ADSC);
		//return 0x00;
}
void BAT_ADC_SENSOR()
{
		ADMUX = 0x60;
		//ADC 3 selected.
		ADMUX |= (1<<MUX0);
		ADMUX |= (1<<MUX1);
		//Start Conversion
		ADCSRA |= (1<<ADSC);
		//TODO CHECK IF BAT IS LOW.


		//return 0x00;
}

unsigned char CALCULATE_MOTOR_SPEED()
{
		float speed = (((sensorData[1]-MOIST_MIN)/(MOIST_MAX-MOIST_MIN))*(MOTOR_MAX_SPEED-
MOTOR_MIN_SPEED))+MOTOR_MIN_SPEED;
		speed = 100.0 - speed;
		unsigned char OCRValue = ((speed*255)/100);
		return ((unsigned char) OCRValue);
}

/*
unsigned char NORMALIZE_VALUE(unsigned char value, unsigned char maxVolt, unsigned char minVolt, unsigned char maxVal, unsigned
char minVal)
{
		unsigned char steps = maxVal-minVal;

		unsigned char minVoltValue = ADC_VALUE(minVolt);

		float resolution = (ADC_VALUE(maxVolt)-minVoltValue)/steps;

		float result = ((value-minVoltValue)/resolution)+minVal;

}
```

```
*/
float ADC_VALUE(float volt)
{

	//if(((int) volt) == ((int) VCC))
	//		return 255.0;

	return (((volt*1024)/VCC)/4);
}

void APPLY_CRC3(unsigned char input)
{
	crc3Var = CRC3(input);
	input &= 0b11100000;
	crc3Var |= input;

	//return packet;
}

unsigned char CRC3(unsigned char input)
{
	unsigned char crc = input;
	int counter;
	for(counter = 0; counter<3; counter++)
	{
		if(crc >= 0x80) //most significant bit is 1.
		crc = crc ^ CRC_POLY;

		crc = (crc<<1);
	}
	crc = crc>>3;

	return crc;
}
unsigned char CRC3_CHECK(unsigned char input) {
	return (CRC3(input) == 0x00);
}

unsigned char current_crc = -1;

unsigned char CRC11(unsigned char data, unsigned char command)
{
	unsigned char crc = data;
	unsigned char packet = command;
	current_crc = crc;
	int counter;
	for(counter = 0; counter<11; counter++)
	{
		if(crc >= 0x80)
			crc = crc ^ CRC_POLY;

		crc = (crc<<1);
		unsigned char temp = (0b10000000 & packet);
		packet = packet<<1;
		crc |= (temp>>7);
	}
	crc = crc>>3;
	current_crc = crc;
	return crc;
}

//REturns command with correct CRC11.
unsigned char APPLY_CRC11(unsigned char data, unsigned char command)
{
	unsigned char crc = CRC11(data, command);
	command &= 0b11100000;
	command |= crc;

	return command;
}
```

## User Code:

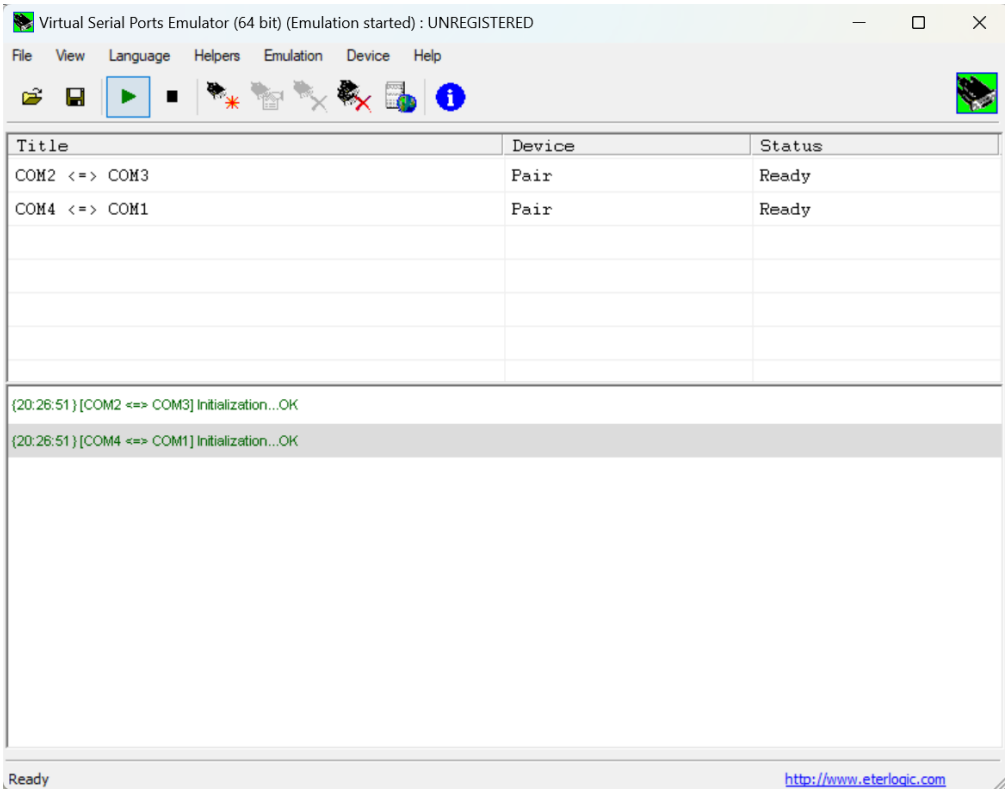
```
/*
 * UserNode.c
 *
 * Created: 26-May-23 20:12:03
 *  Author: msoma
 */
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <avr/wdt.h>
#include <avr/eeprom.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <util/delay.h>


#define F_CPU 8000000

#define LCD_2_LINE_MODE 0x38
#define DISPLAY_ON 0x08
#define CLEAR_DISPLAY 0x01
#define SHIFT_AFTER_DISPLAY 0x06
#define DISPLAY_BLINK 0x0C
#define DISPLAY_ON_BLINK 0x0F

void PORTS_INIT();
void USART_INIT();
void LCD_INIT();
void sendCommand(unsigned char command);
void sendData(unsigned char data);
void sendLCD(unsigned char isCommand, unsigned input);

void CHECK_INPUT();
void TRANSMIT(unsigned char message);

void SLEEP();
void CLEAR_VARS();

unsigned char totalChars = 0;
unsigned char toClear = 0;
unsigned char lastChar = 'a';

unsigned char keypad[4][3] = { {'1', '2', '3'},
                               {'4', '5', '6'},
                               {'7', '8', '9'},
                               {'.', '0', '.'}};

unsigned char interuptCount = 0;
unsigned char isStarted = 0;
unsigned char dataSent = 0x00;

int main(void)
{
	PORTS_INIT();
	LCD_INIT();
	USART_INIT();

	//Enable INT1.
	//GICR = 0x80;
	//Enable ISC11 and ISC10. Rising edge on INT 1.
	EICRA = 0x0C;
	EIMSK = 0x02;

	//sendData('R');
	//sendData('E');
	//sendData('S');
```

```
	//lastChar = '\0';
	//sendData('\0');


	isStarted = 1;
	//sei();
	while (1)
	{
		SLEEP();
		//CHECK_INPUT();
	}
}



void PORTS_INIT()
{
	/*
	DDRA.0 = RS
	DDRA.1 = RW
	DDRA.2 = E
	*/
	DDRA = 0xFF;

	//DATA output.
	DDRB = 0xFF;

	//Keyboard
	DDRC = 0xF0;
	PORTC = 0x07; // ground all rows CHIGH
}

void USART_INIT()
{
	UCSR0B = (1<<TXEN0 | 1<<TXCIE0 | 1<<RXEN0 | 1<<RXCIE0);
	UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);
	UBRR0H = 0x00;
	UBRR0L = 0x33;
}
void LCD_INIT()
{
	CLEAR_VARS();
	//2 Line display:
	//https://morf.lv/simple-library-for-driving-20x4-lcd-with-4bits
	sendCommand(LCD_2_LINE_MODE);
	sendCommand(DISPLAY_ON_BLINK);
	sendCommand(CLEAR_DISPLAY);
	sendCommand(SHIFT_AFTER_DISPLAY);
}

void CHECK_INPUT()
{
	unsigned char col = -1;
	unsigned char row = -1;
	unsigned char i = 0;

	for(i=0; i<10; i++)
	{
		_delay_ms(20); // for debounce
		col = (PINC & 0x07); // read columns

		if(col != 0x07)
			break;
	}
	if(col == 0x07)
		return;

	for(i=0; i<10; i++)
	//while(1)
	{
		PORTC = 0xEF; // ground row 0
```

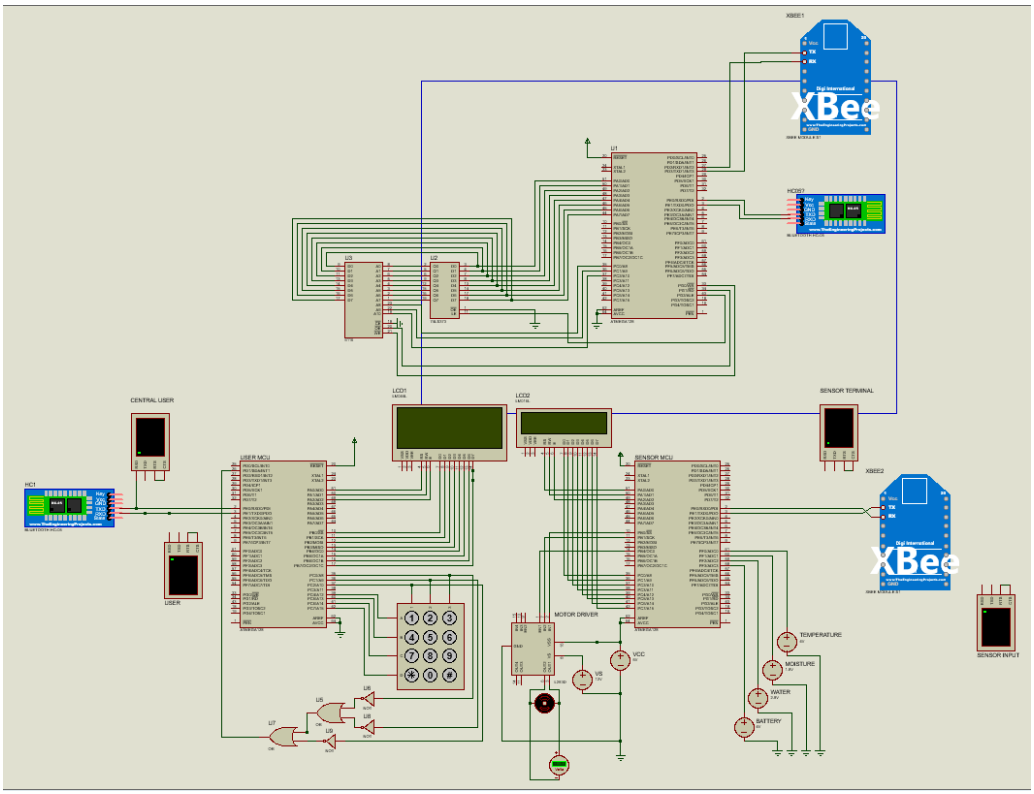
```
                col = (PINC & 0x07); // read columns
                if (col != 0x07) // column detected
                {
                        row = 0; // save row location
                        break; // exit while loop
                }

                PORTC = 0xDF; // ground row 1
                col = (PINC & 0x07); // read columns
                if (col != 0x07) // column detected
                {
                        row = 1; // save row location
                        break; // exit while loop
                }
                PORTC = 0xBF; // ground row 2
                col = (PINC & 0x07); // read columns
                if (col != 0x07) // column detected
                {
                        row = 2; // save row location
                        break; // exit while loop
                }
                PORTC = 0x7F; // ground row 3
                col = (PINC & 0x07); // read columns
                if (col != 0x07) // column detected
                {
                        row = 3; // save row location
                        break; // exit while loop
                }
        }               unsigned char toOut;            if (col == 0x03)
                toOut = (keypad[row][0]);
        else if (col == 0x5)
                toOut = (keypad[row][1]);
        else //(col == 0x06)
                toOut = (keypad[row][2]);

        //interuptCount=0;
        PORTC = 0x07;
        //interuptCount=1;
        sendData(toOut);
        TRANSMIT(toOut);
}

void TRANSMIT(unsigned char message)
{
        dataSent = message;
        UDR0 = message;
}

void sendCommand(unsigned char command) {
        sendLCD(0, command);
}

void sendData(unsigned char data)
{
        if((totalChars%20) == 0 && data == ' ')
                return;

        sendLCD(1, data);
        totalChars++;

        unsigned char lineNum = totalChars/20;
        unsigned char currentChars = totalChars%20;
        unsigned char remainingChars = 20-currentChars;

        if(data == '\r' && currentChars != 0)
        {
                totalChars += remainingChars;
                lineNum++;
        }

        if(remainingChars==20 || data == '\r')
```

```
		{
			if(lineNum == 1)
				sendCommand(0xC0);
			else if(lineNum == 2)
				sendCommand(0x94);
			else if(lineNum == 3)
				sendCommand(0xD4);
			else if(lineNum == 4)
				LCD_INIT();
		}

		//lastChar = data;
}

void sendLCD(unsigned char isData, unsigned input)
{
		PORTA = 0x00+isData;
		PORTB = input;

		PORTA = 0x04+isData;
		_delay_ms(1);
		PORTA = 0x00+isData;
		_delay_ms(1);
}


//Interupt for input
ISR(INT1_vect)
{
		sleep_disable();
		CHECK_INPUT();
}


ISR(USART0_TX_vect)
{
		sleep_disable();

		if(dataSent == '.')
			LCD_INIT();
}

// Sample Interrupt handler for USART0 (User side) when an RX is done
ISR(USART0_RX_vect)
{
		sleep_disable();

		while (! (UCSR0A & (1<<RXC0))){}; // Double checking	flag

		unsigned char dataToSend = UDR0;

		if(lastChar==';')
			LCD_INIT();

		lastChar = dataToSend;
		sendData(dataToSend);
}


void SLEEP()
{
		sleep_enable(); // arm sleep mode
		sei(); // global interrupt enable
		sleep_cpu(); // put CPU to sleep
}

void CLEAR_VARS()
{
		totalChars = 0;
		toClear = 0;
		lastChar = 'a';}
```

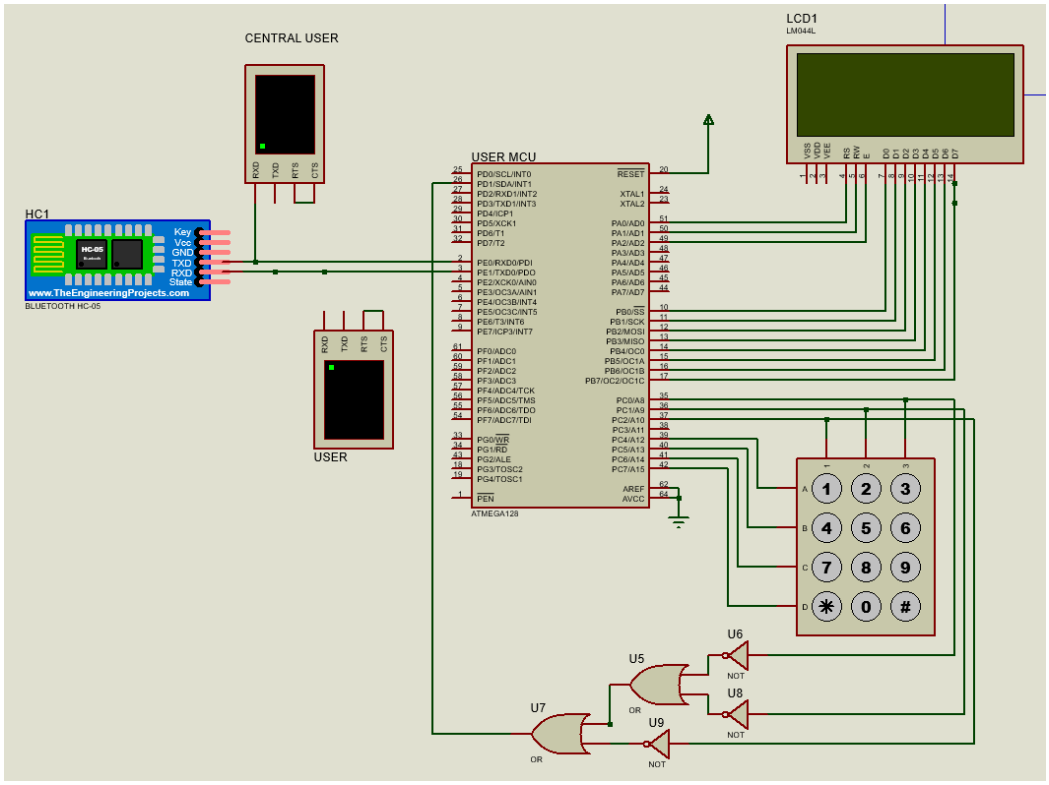
# Verification



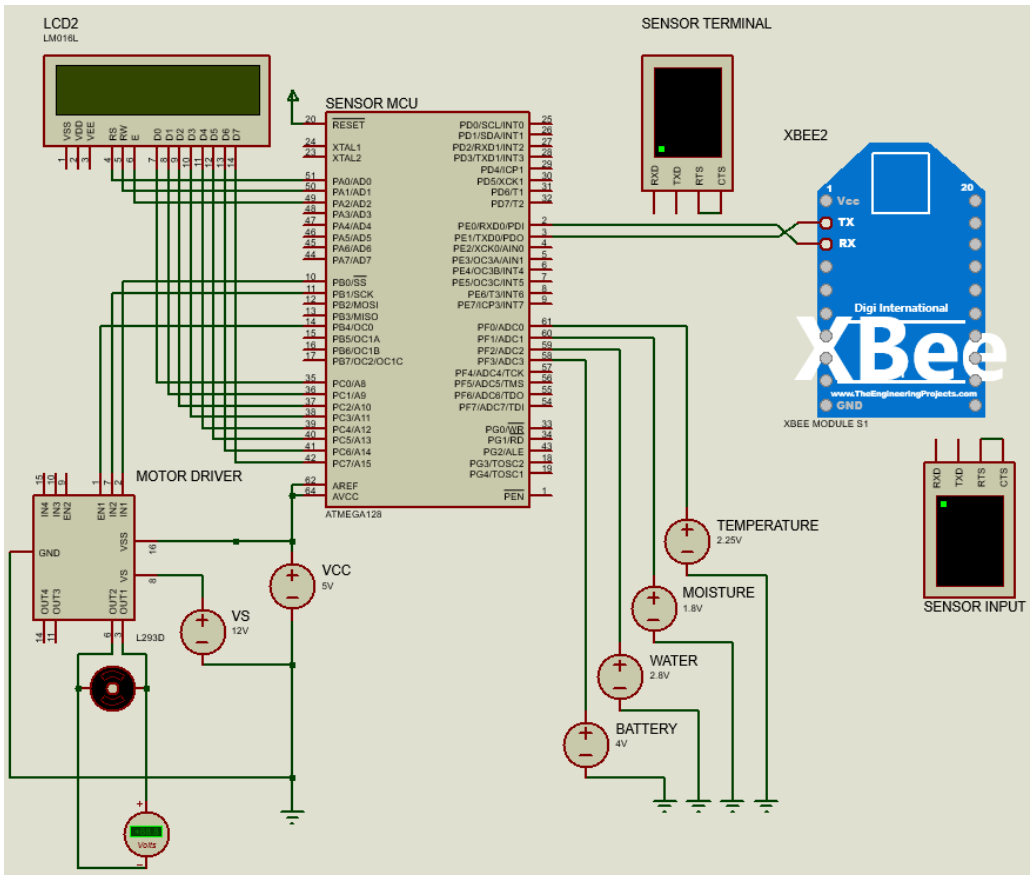

VSPEE Setup is shown in the picture above. COM1 and COM4 are selected for the XBee module pair connecting the central MCU with the sensor MCU assembly. COM2 and COM3 are selected for the Bluetooth module pair connecting the central MCU with the user MCU assembly.



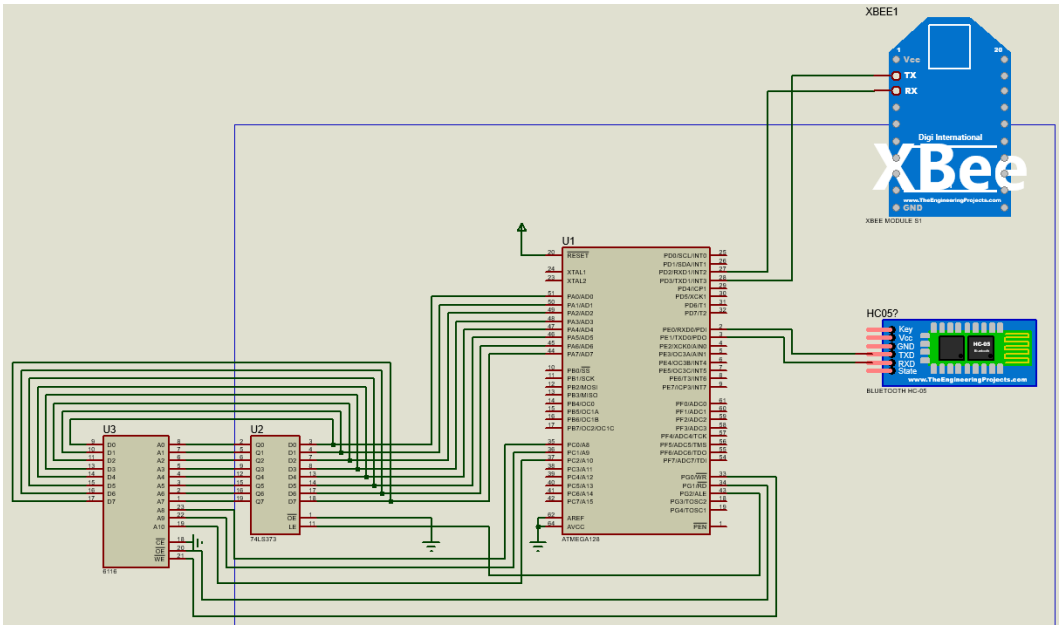
This is the proteus assembly in its entirety. All connections and devices are shown clearly.




This is the user side MCU schematic shown in a magnified view for clarity.

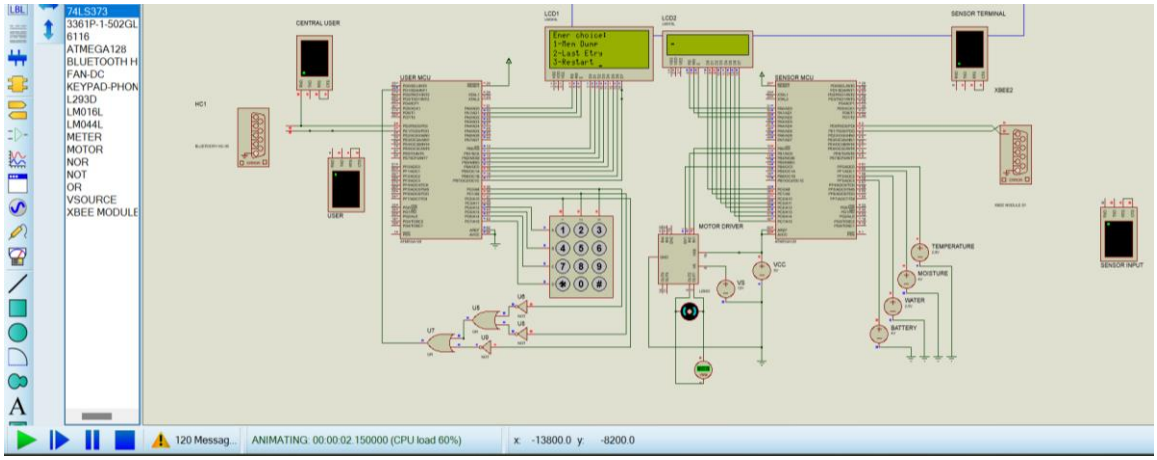



This is sensor side MCU schematic shown in magnified view for clarity.

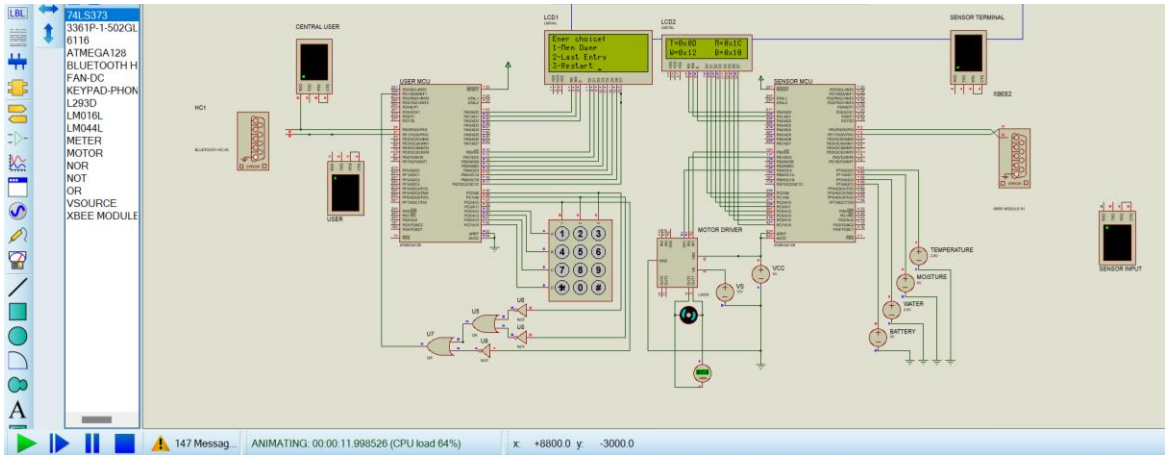



This is central MCU schematic shown in magnified view for clarity.

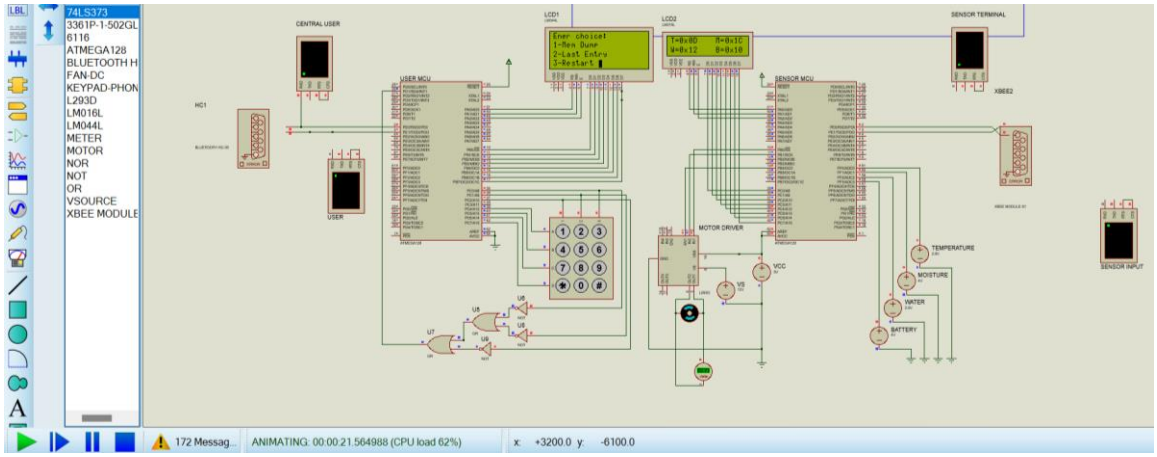
## Sensor LCD Values and Motor Function:




At this moment, the ADC values are not read and the terminal is empty. This is because the values are read after every 10 seconds. The motor is also stopped at this moment.

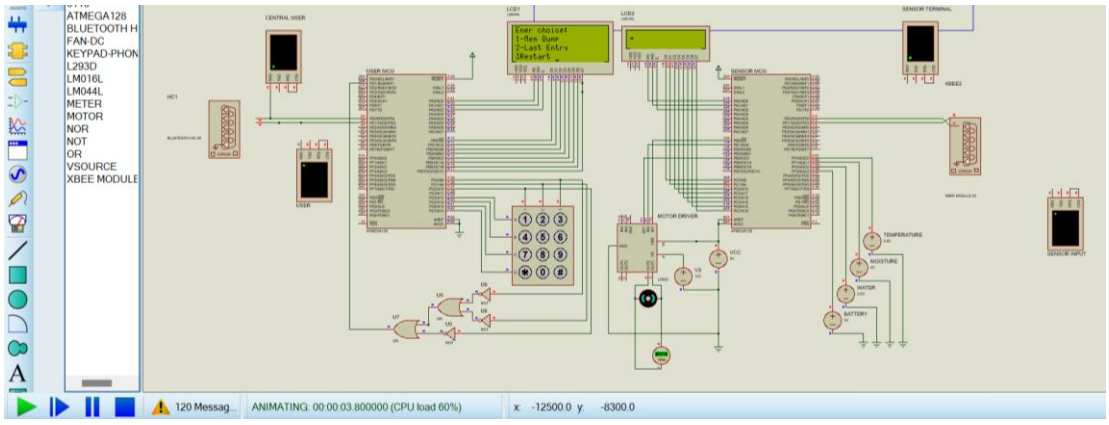



As 10 seconds pass, the ADC parameters are read and displayed into the screen. The motor also starts spinning where the speed is dependent on the moisture level value.

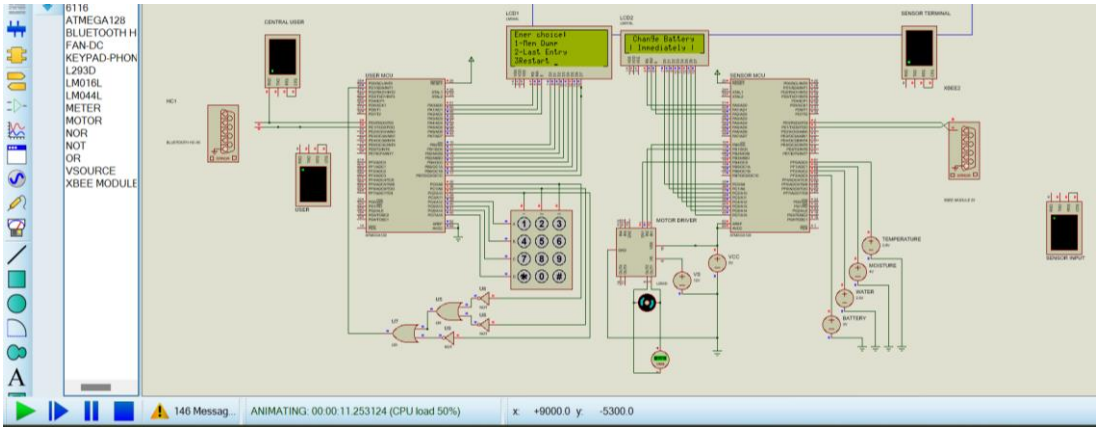



As can be seen in this screenshot, the motor is spinning. Since while running the simulation, the input voltages to the sensors can not be changes, the LCD displays the same values every 10 seconds. The motor spins for 5 seconds and then stops but this cannot be shown in static pictures. We will show this during our demo.

## **Battery:**

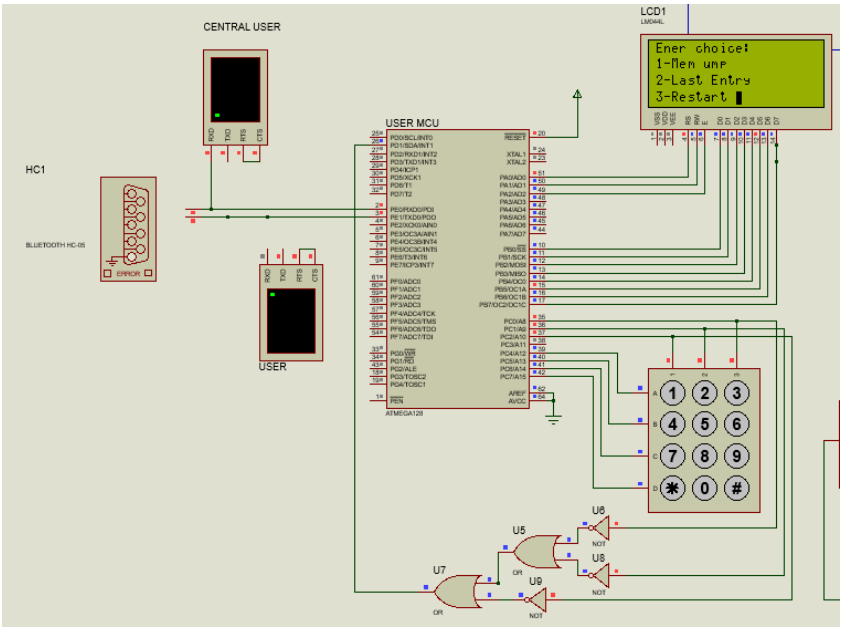



In this case, battery is set as 3V but the time is less than 10 seconds so there is no display in the sensor LCD.

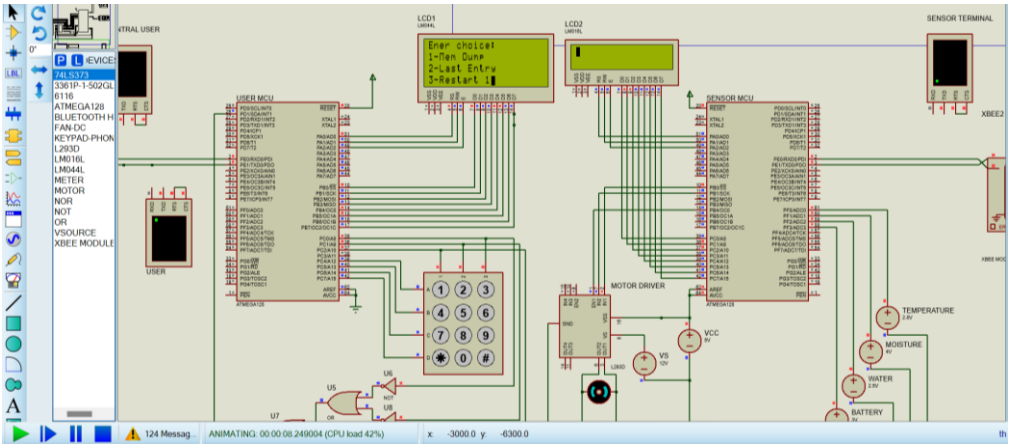



After 10 seconds pass, “change battery immediately” message is displayed.

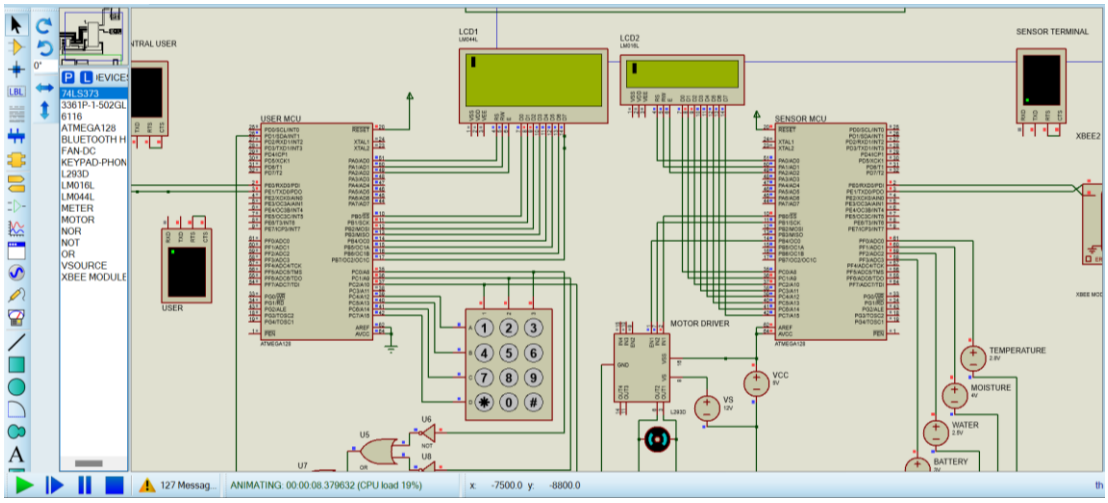
## User LCD Display:

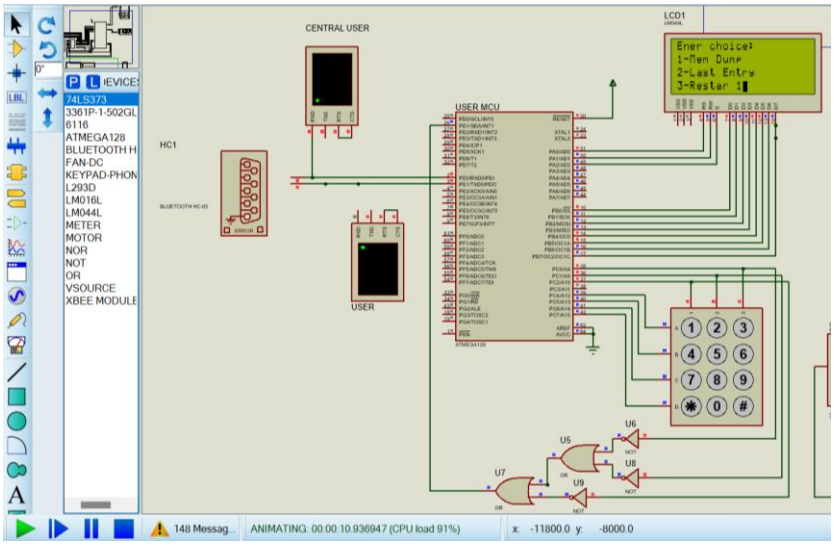



In the picture above, the sensor side is shown with user menu displayed to ask the user for input. Depending on the input, all data stored in memory can be displayed, or the last entry can be displayed, or the system can be reset.

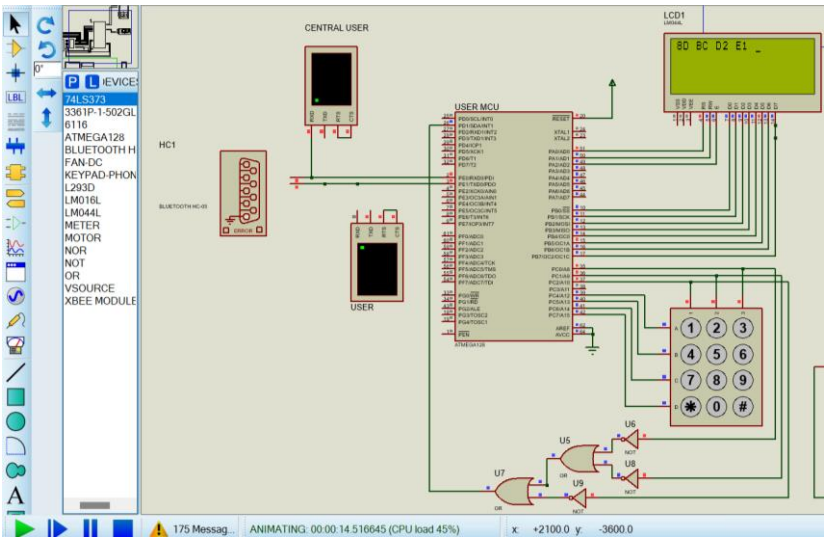


Initially there is not data in the system. When we enter “1.” to memory dump, we get no display as shown in the next picture.

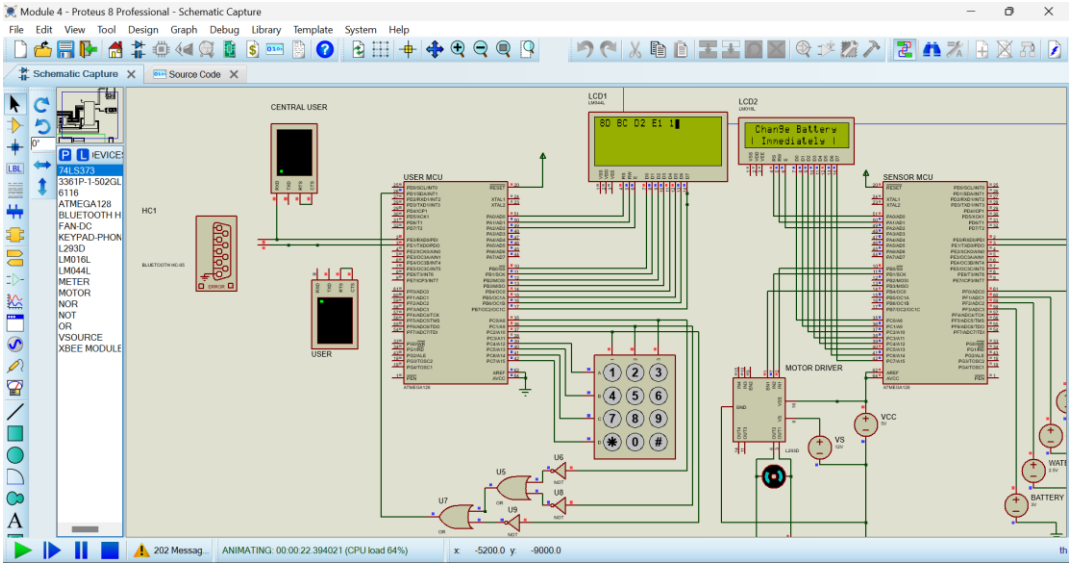


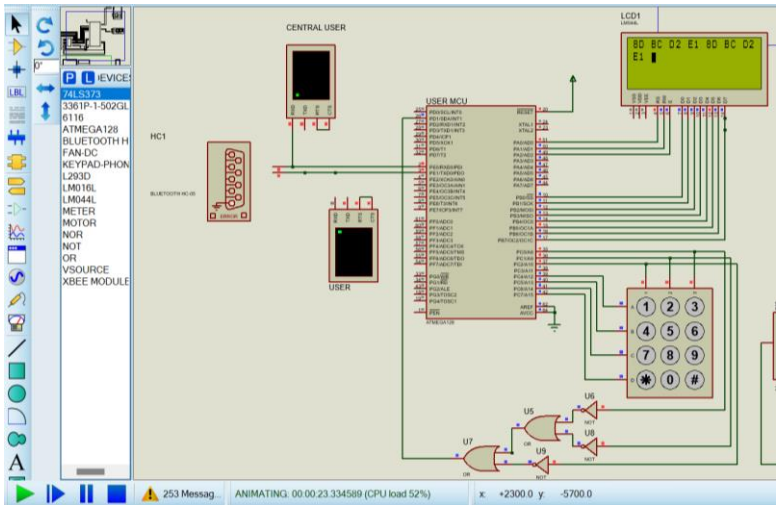

Initially there is not data in the system so we wait for 10 seconds for the sensors to read the value. Then we enter “1.” to order memory dump.



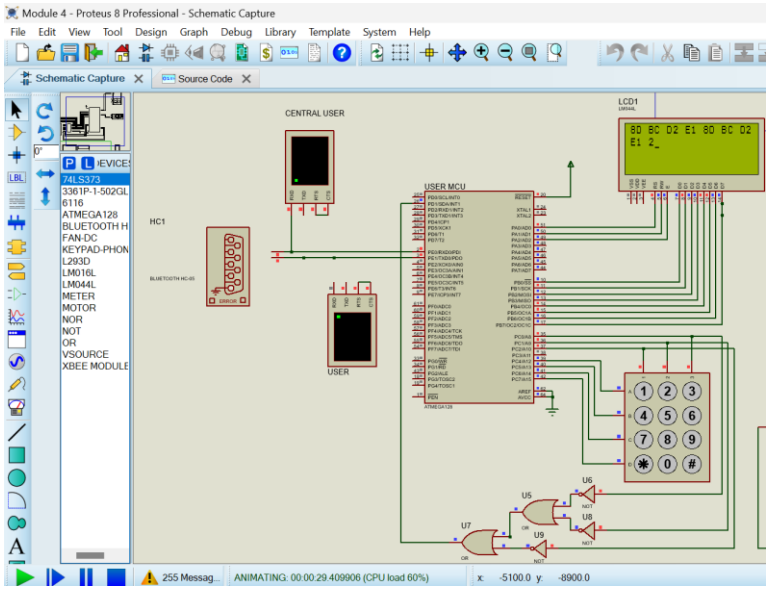

As seen in the terminal, all values logged in the memory are displayed.



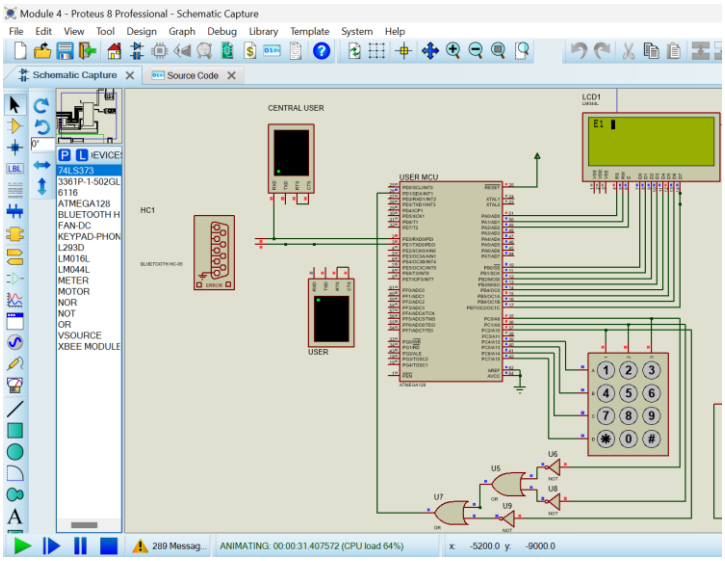

After waiting 10 more seconds, memory dump command is entered again.



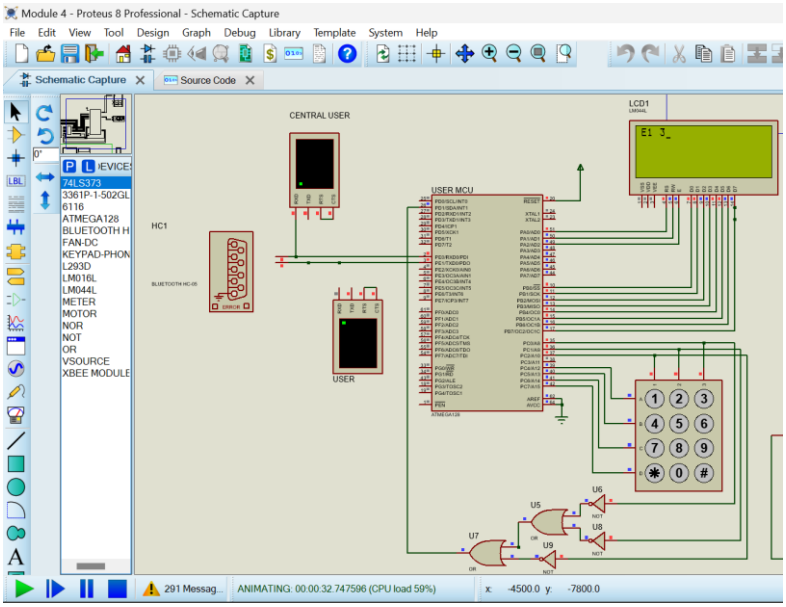

As seen in the LCD terminal, 8 values are now displayed after waiting 20 seconds. This is as expected since the sensors record 4 values at a time every 10 seconds.



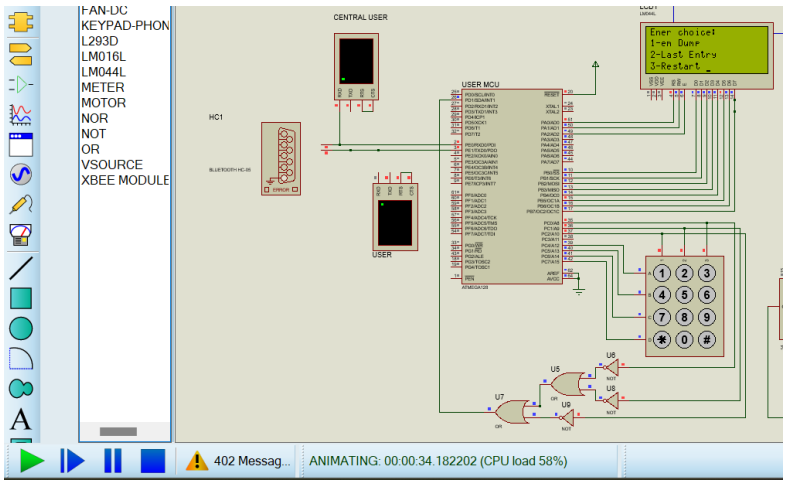

Now ordering last entry by entering “2.”.



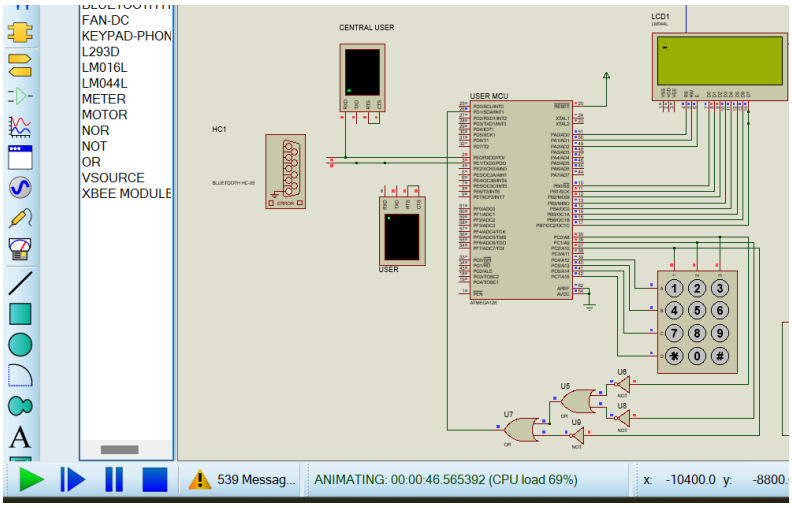

As seen in the picture above, last entry is displayed correctly.




Now ordering reset by entering “3.”.



It is seen that the system is reset.




Now requesting last entry or memory dump results in null value since the system is reset.

# Conclusion

Overall, this was a very good project. It enabled us to improve our problem solving skills. We learned how to implement MCU's especially atmega128 technology to design control systems.

**Motor:** 1 W * 28800s = 28800 J of energy is consumed by the motor in a 24 hour period.

**16x2 LCD:** (6/1000)W*28800s = 172.8 J of energy is consumed by the 16x2 LCD.

**20x4 LCD:** (17.5/1000)W*28800s = 504 J of energy is consumed by the 20x4 LCD.

**HC-05 Bluetooth:** for no communication, the voltage is 3.3 V but the current is 8mA for no communication and 20mA while communicating. This gives the best case power dissipation as 26.4mW and worst case as 66mW.

**XBee:** for transmit and receive, the voltage is 3.3 V but the current is 45mA and 50 mA respectively. This gives the best case power dissipation as 148.5mW and worst case as 165mW.

**MCU (central):** the voltage is 5V and the current in the best case (sleep mode) is 8mA and the current in the worst case is 17mA. The power consumption in the best case is 40mW and the worst case is 85mW.

**MCU (user-node):** the voltage is 5V and the current in the best case (sleep mode) is 8mA and the current in the worst case is 17mA. The power consumption in the best case is 40mW and the worst case is 85mW.

**MCU (remote node):** the voltage is 5V and the current in the best case (sleep mode) is 8mA and the current in the worst case is assumed as 25mA. The power consumption in the best case is 40mW and the worst case is 125mW.