MIDDLE EAST TECHNICAL UNIVERSITY NORTHERN CYPRUS CAMPUS
ELECTRIC AND ELECTRONIC ENGINEERING PROGRAM

EEE-347 LAB #4

FINAL-REPORT

Working with Standard User Interface Devices,

A/D Converters, and Motors

Muhammed Emre Duman-2079754

Muhammet Talha Erikel-2079820

OBJECTIVES:

In this laboratory module , it is aimed integration by using standard user input interfaces with sensor signals which are processed by ATmega A/D converter, and a DC water-pump (motor) to build the smart farming system. In addition to previous lab module, it is added some new features. User-side display ,user-side keypad, Remote node sensors , remote node instant display and remote – node water pump (DC motor) are the new features for this laboratory module.

This project has been designed by Talha Erikel and Emre Duman. Each segments of this project have been done by two member. Thus, whole project is double-checked.

4.3 DESIGN AND REPORTING

4.3.1 Preliminary Work

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.

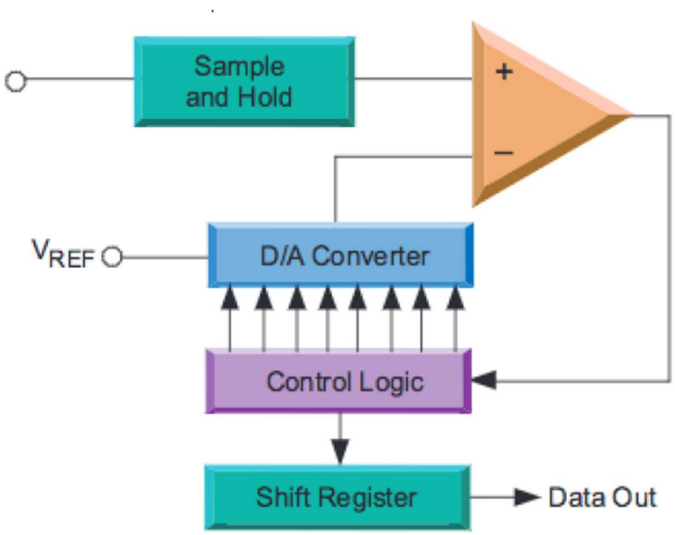


Figure 4.2. Successive Approximation ADC

➔

Sample and hold circuit samples and stores analog input Vin. *Control logic* generates and sends a mid-range digital value to the *D/A Converter*. According to *V(ref)*, *D/A converter* converts the a mid-range digital value to the analog signal. *Op-amp compares* two analog signals. Op-amp sends the comparison to *the control logic and control logic decides* the next digital value. The system works until 8-bit has done. End of the conversion, *shift register* is going to shift all data to the data out by 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.

Normal conversion takes 13 clock cycles. To stay under 200kHz, must divide 16MHz clock by 128, giving 125kHz ADC clock. For 13 cycle conversion -> 13*(1/125000)=104 us per conversion.
ADCSRA = (1<<ADEN) | (1<<ADFR) | (1<<ADPS2)| (1<<ADPS1)| (1<<ADPS0); // free-running mode, 128 prescale

ADMUX = (1<<REFS0)| (1<<ADLAR) ; // AVCC REF, 8-bit, left justified

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.

Parameter	Min. (V)	Max. (V)	Min. (digital)	Max. (digital)	Eff. Resolution (mV)
T	2.0	4.0	0x03	0x1B	$0.083\ V$
M	1.8	4.2	0x02	0x1E	$0.05\ V$
W	2.0	2.8	0x04	0x1A	$0.036\ V$
B	3.0	5.0	0x01	0x01F	$0.067\ V$

$T\ Calculation$:
$Range\ of\ digital = 0x1B - 0x03 = 0x18$
$Range\ of\ analog = 4 - 2 = 2\ V$
$\frac{2V}{24} = 0.083\ V$

$M\ Calculation$:
$Range\ of\ digital = 0x1E - 0x02 = 0x1C$
$Range\ of\ analog = 4.2 - 1.8 = 1.4\ V$
$\frac{1.4V}{28} = 0.05\ V$

$W\ calculation$:
$Range\ of\ digital = 0x1A - 0x04 = 0x16$
$Range\ of\ analog = 2.8 - 2 = 0.8V$
$\frac{0.8V}{22} = 0.036\ V$

$B\ calculation$:
$Range\ of\ digital = 0x01F - 0x01 = 0x01E$
$Range\ of\ analog = 5 - 3 = 2V$
$\frac{2V}{30} = 0.067\ V$

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?

The PWM generation can be set by adjusting duty cycles. Therefore, there are four settings for the water pump speed.

The clock 0 is going to be set as a “PWM phase correct”, and non-inverting mode with no prescale.

TCCR0=0x61;

Duty cycles (100%) -> OCR0=255

Duty cycles (75%) -> OCR0=191

Duty cycles (50%) -> OCR0=127

Duty cycles (25%) -> OCR0=63

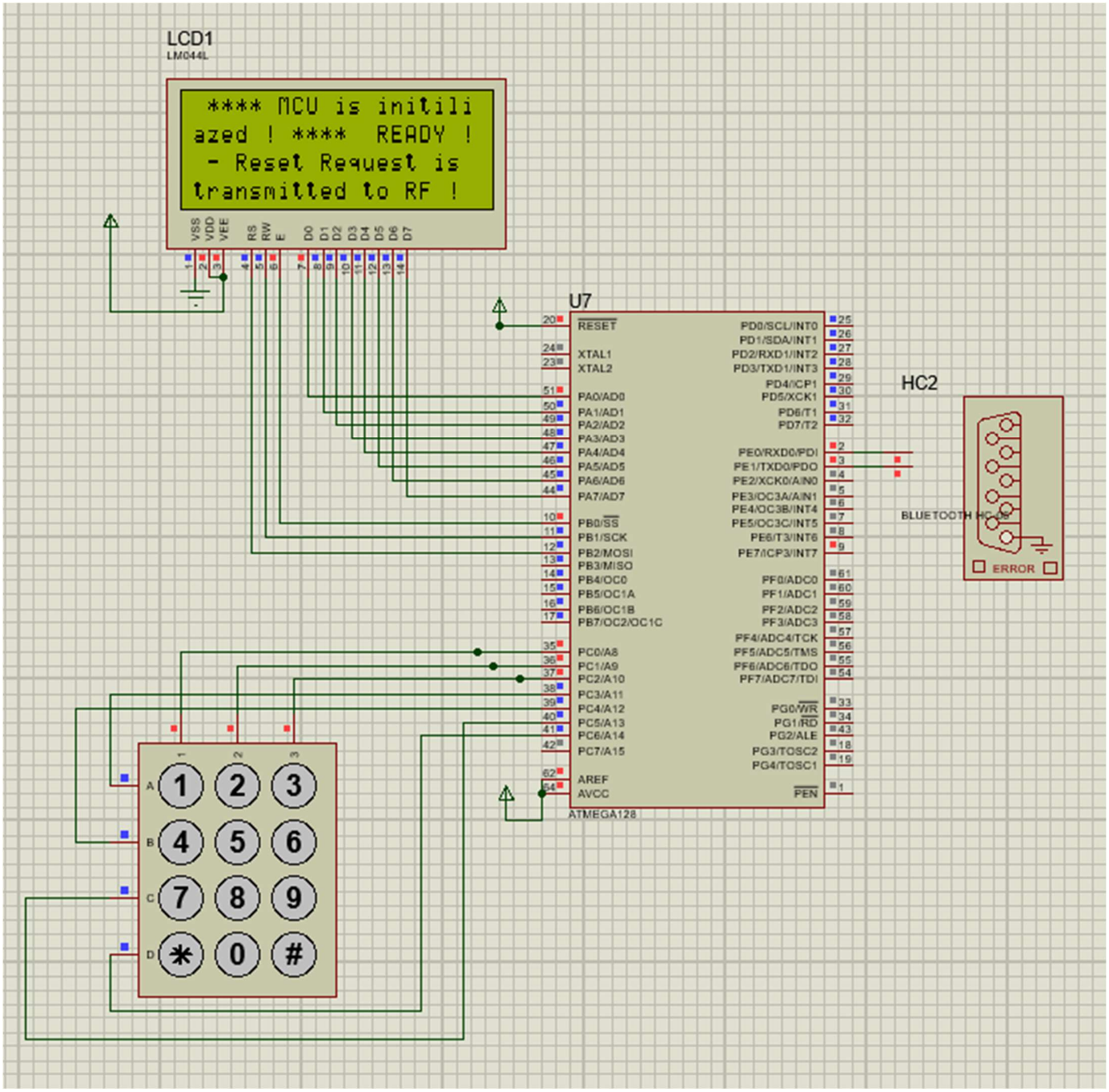
Since the motor speed is adjusted for moister level , default motor speed should be 0. Moreover, normally, motor does not run, it runs when the MCU sense the moister level.

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

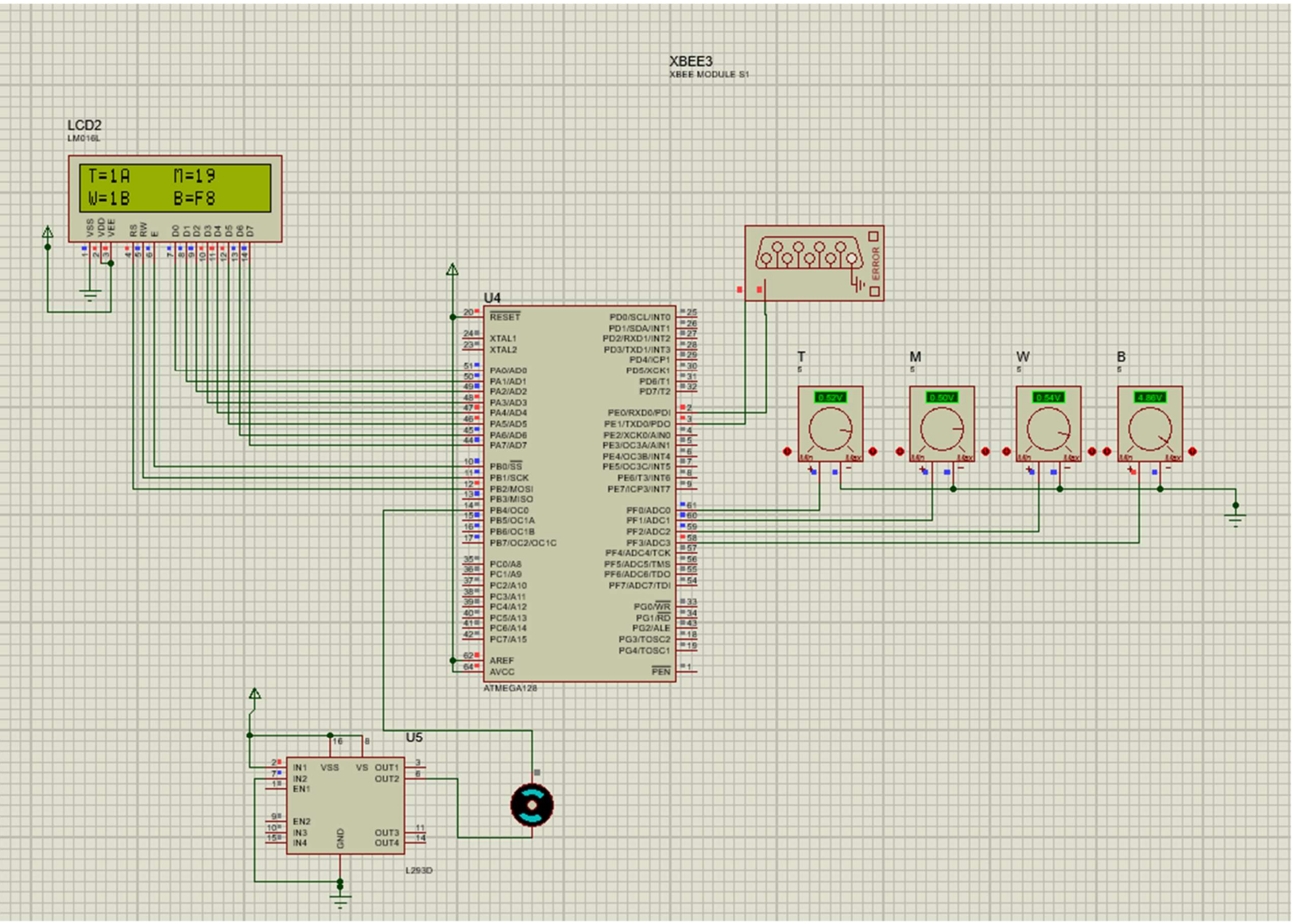
Utilizations of 16x2 LCD and 20x4 LCD are nearly same. However, 16x2 LCD is able to display 16 character each two line, so the cursor position function should modify for this manner. 20x4 LCD's cursor position function must modify in this manner as well.

f) Sketch a system schematic diagram that has the full smart farming system, including 3 ATmega128 MCUs and their connectivity to the periperhal 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.

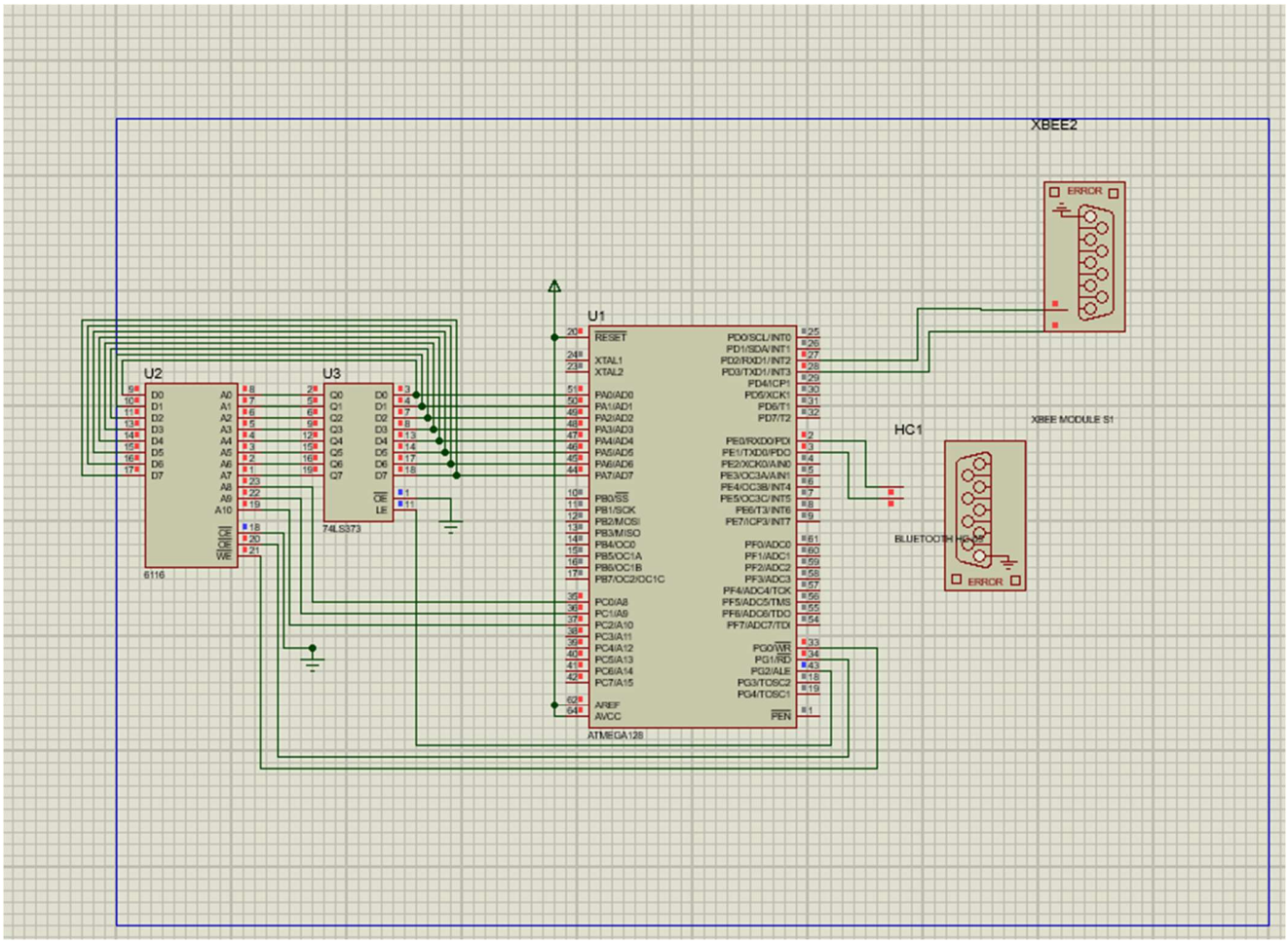
User interface Side :



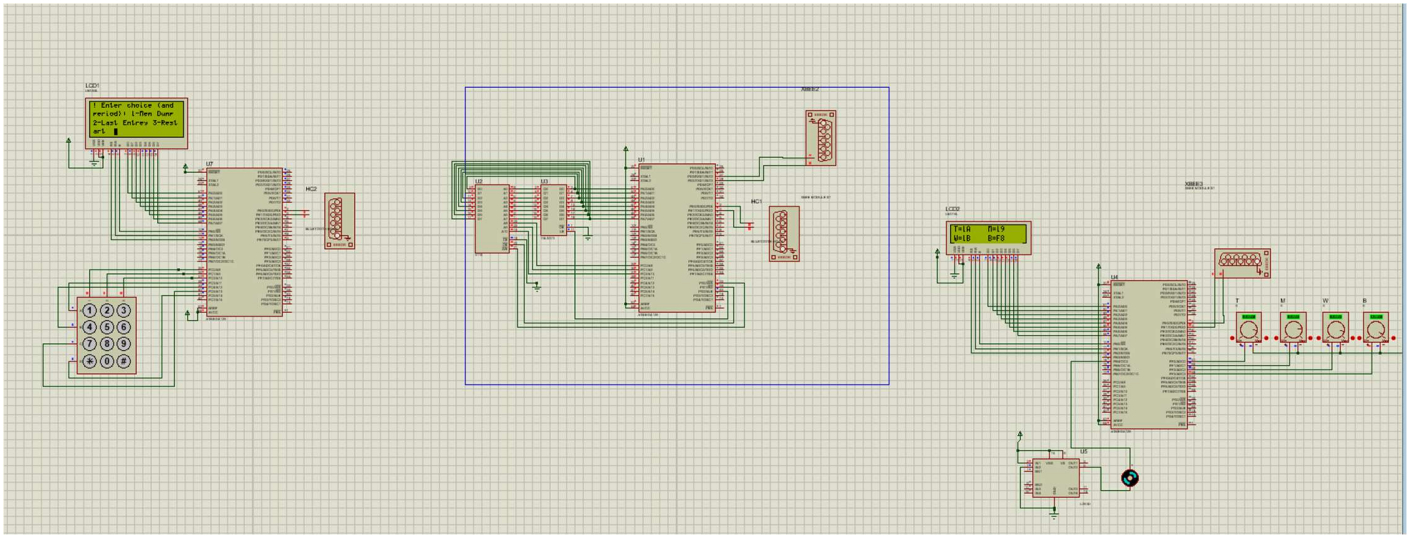
Remote Control Side:



General CPU from lab module 3:

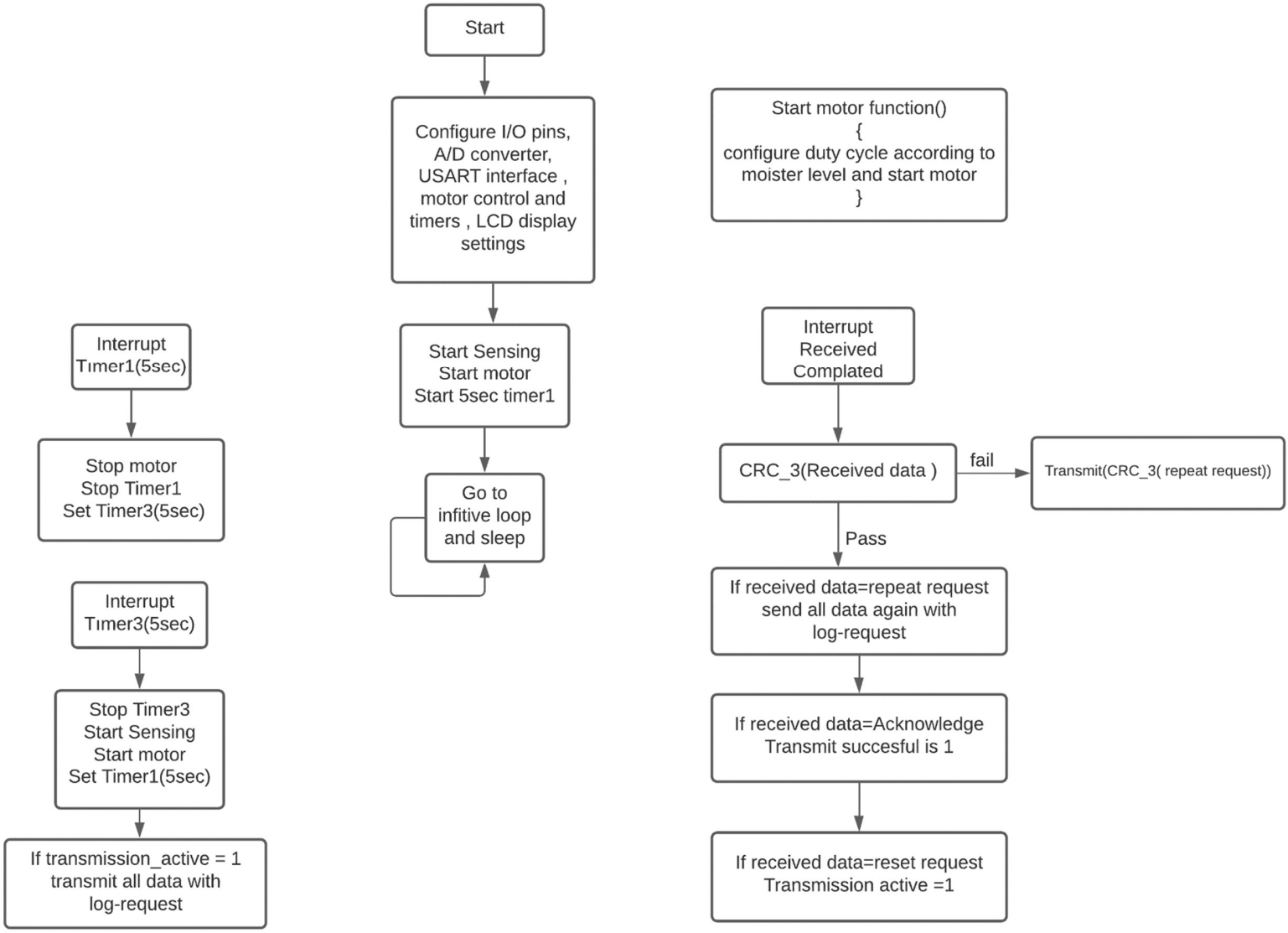


The full smart farming system:

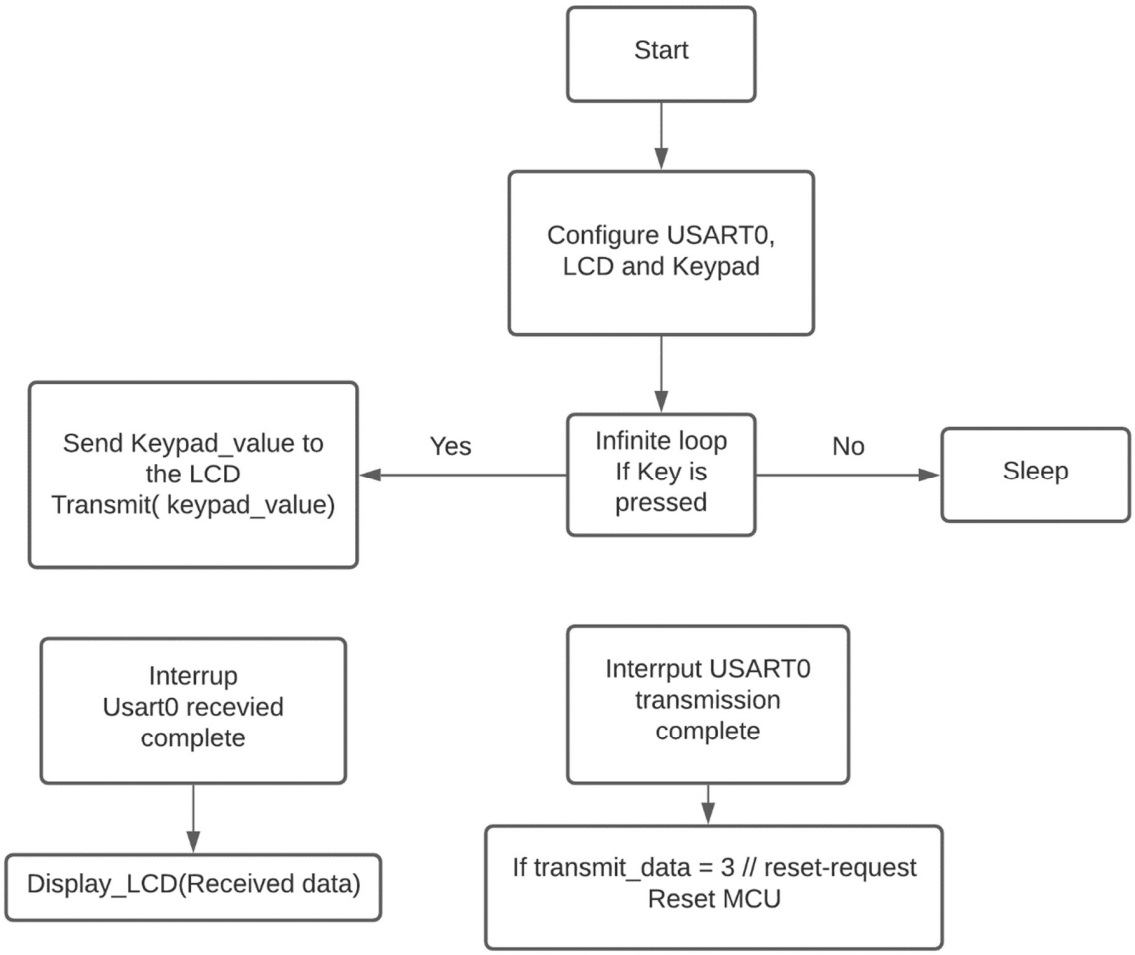


All schematics are shown above in an order to seen clearly all ports .

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



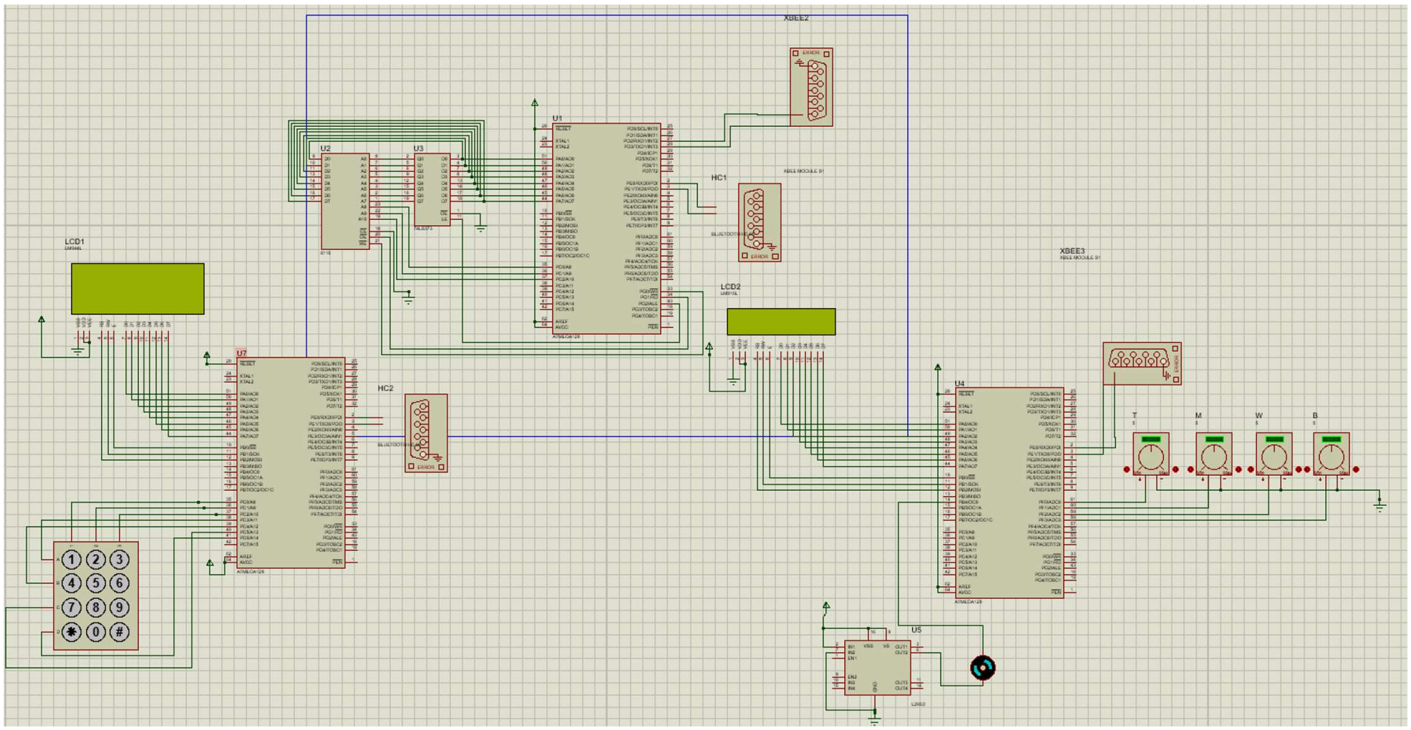
h) Sketch an algorithmic flowchart to accurately show the program executed in the User Node MCU.



i) Considering your answers to (f-h), and sytem 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.

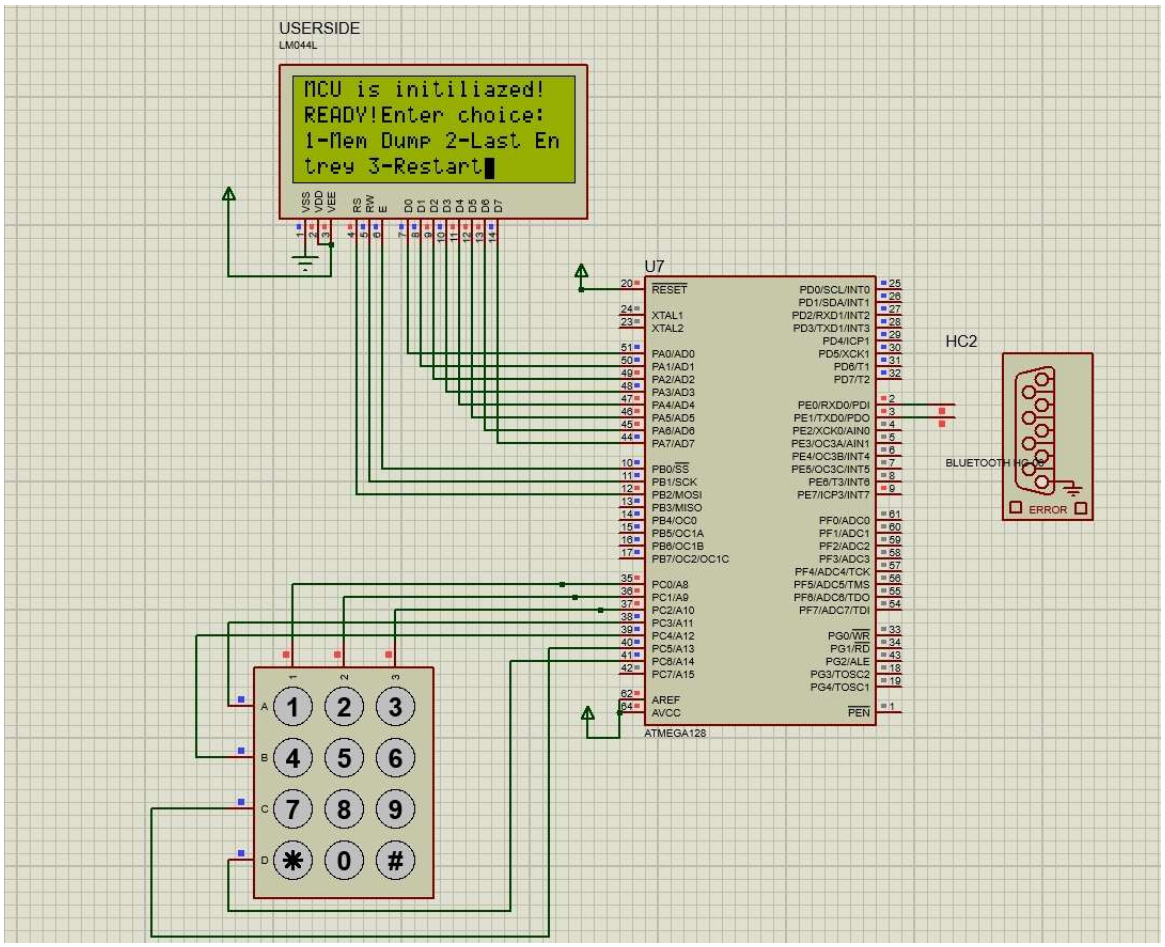
Component Power (mW)	Approx. best-Case (IDLE)	Approx. worst-Case (ACTIVE)
MCU (Central)		
MCU (User-node)		
MCU (Remote-node)		
Bluetooth Interface		
Xbee Interface	50mA (@ 3.3 V)	
16x2 LCD display		
20x4 LCD display		
Motor driver	0	200
Waterpump	0	1000

4.3.3 Verification



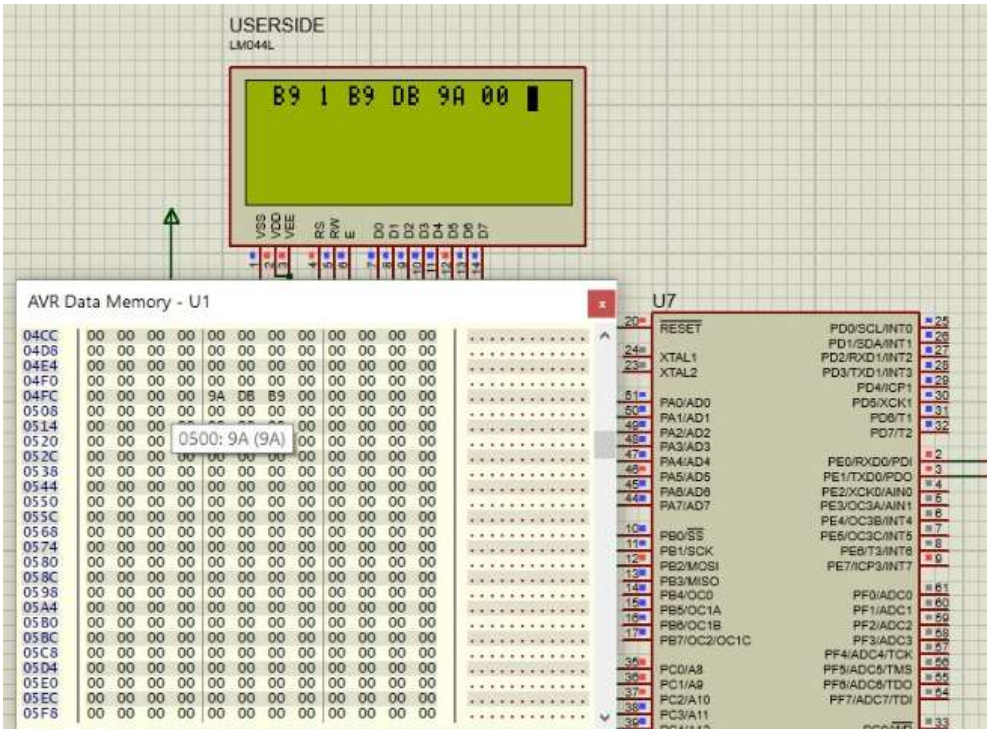
There are 3 MCU in the smart farming system, and the schematic shows all MCUs at the same time. At the left side of the schematic represent user side interface and right side of the schematic represent the remote sensor interface. The middle MCU, which is used on lab module 3 is also shown .

User Side interface :



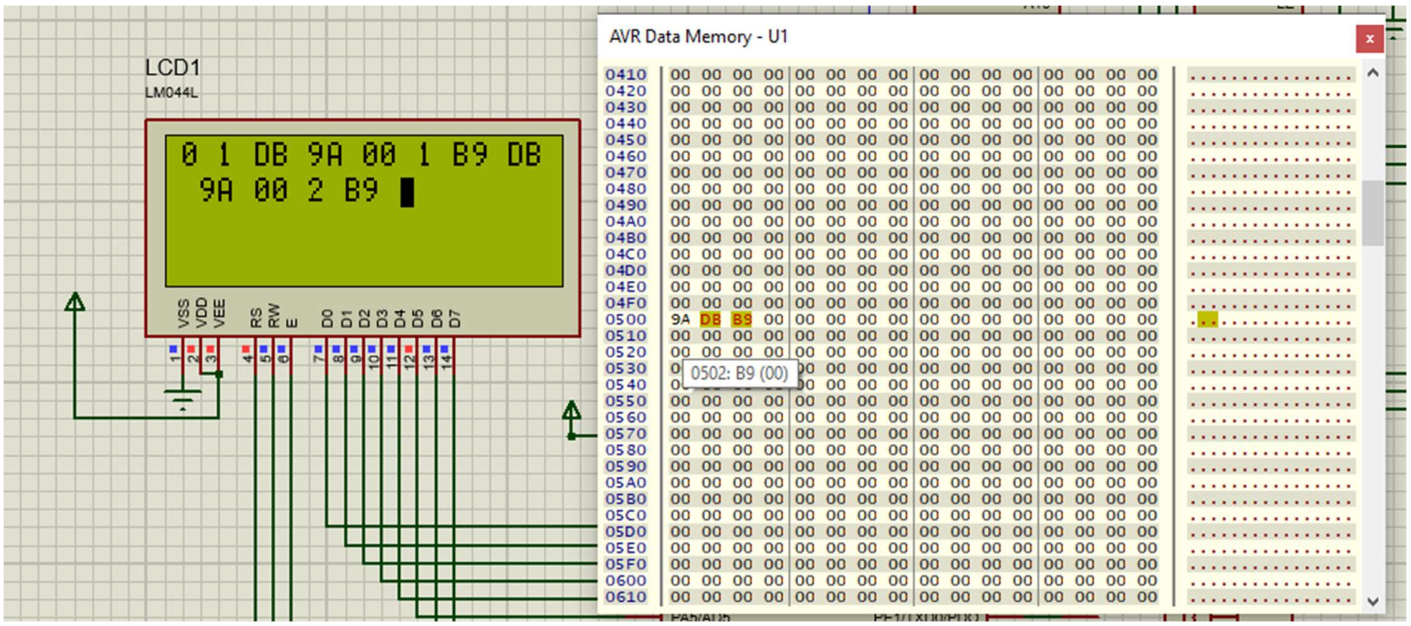
- The battery level parameter is excluded from simulation due simplicity.

➔ This picture represents user side interface at the first run . LCD displays the strings which is taken from general MCU, and system ready to take input from keypad.

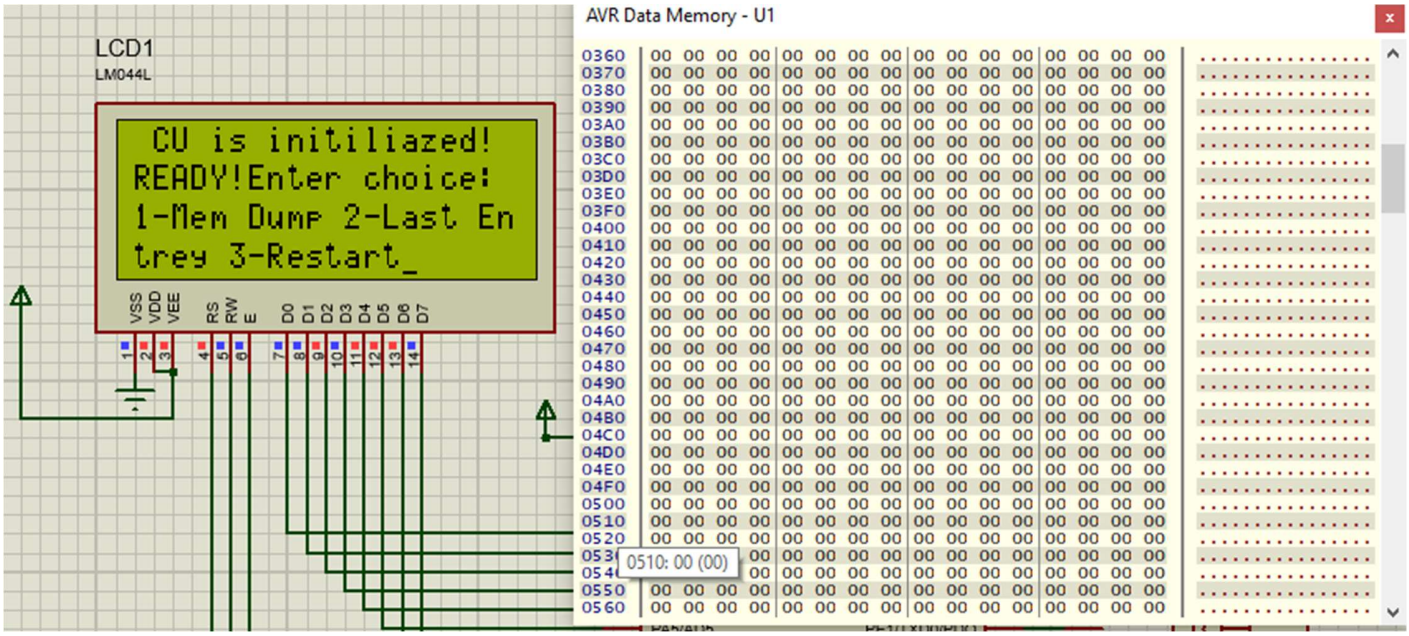


➔ The remote sensor side sent three different parameters to the general MCU. Then, the user side sent a memory damp request by pressing “3” on the keypad. After that , general MCU sent back to the user side three saved parameters. Finally, the user side display all saved data to the LCD.

*The AVR data memory window (general MCU) is shown left.

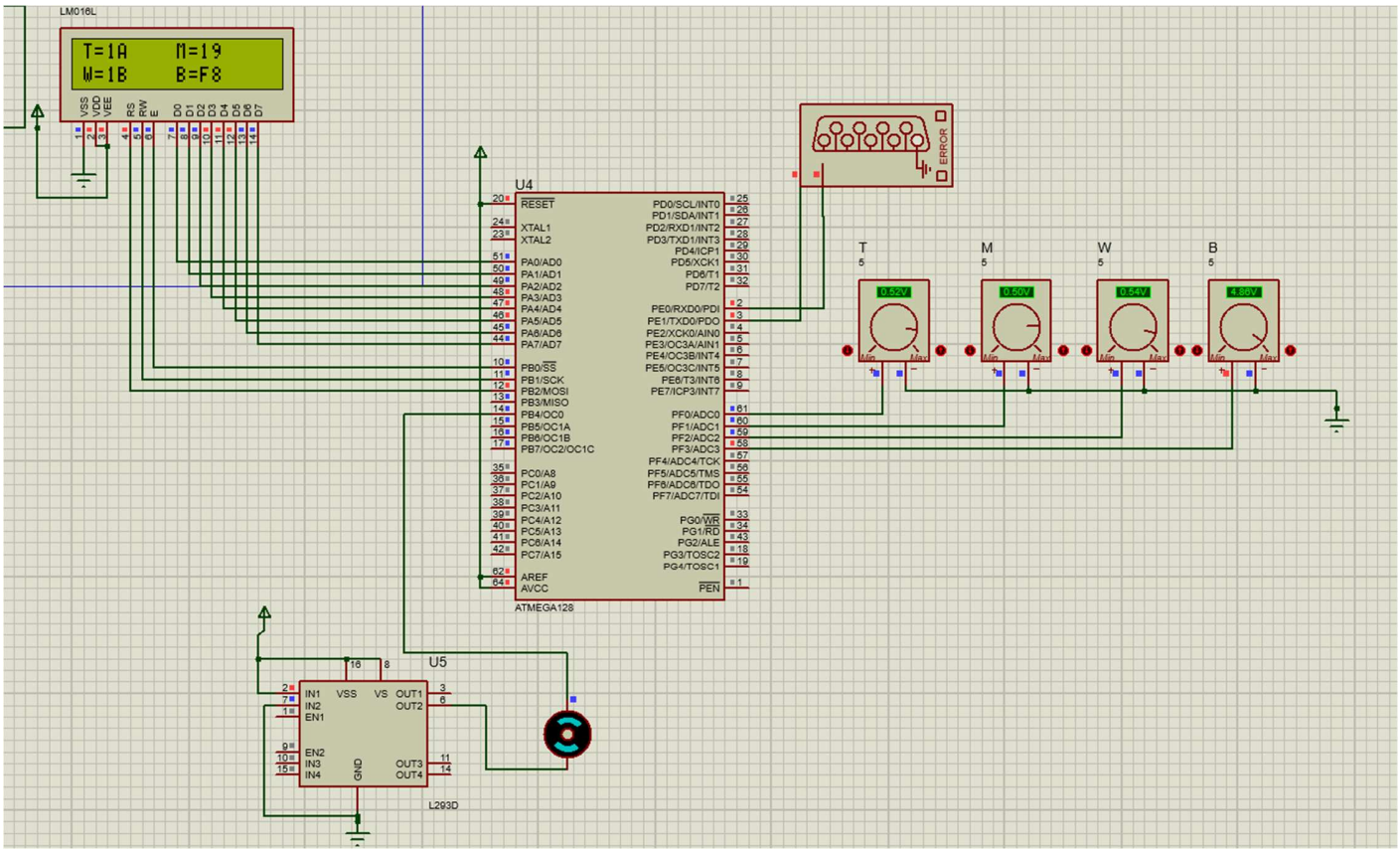


➔ The same scenario is applied with upper case. However, in this time , from the user side , two (2) is pressed on the Keypad, therefore; the general MCU received last entry request, so user side MCU displays the last saved data.



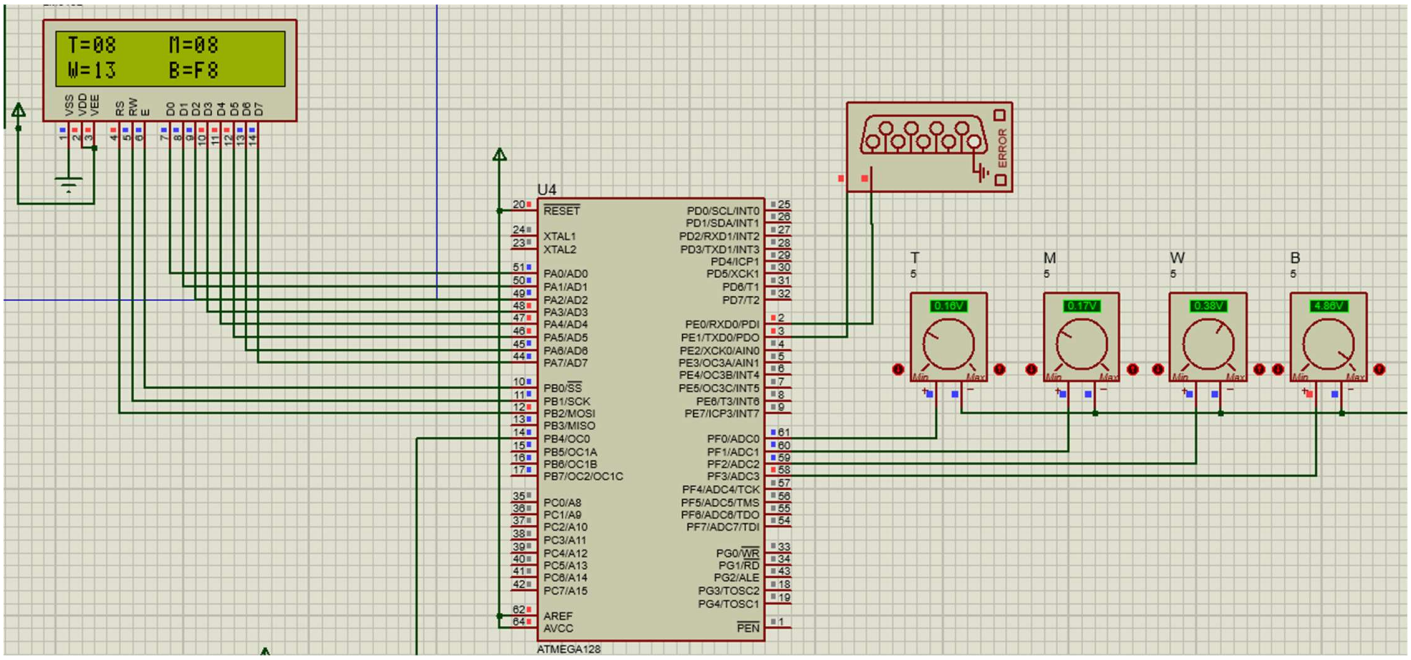
In this situation, it is pressed “3” for the restart request. The program has worked as it is expected and restart itself. After restart operation, program has been stopped to see inside of the AVR data memory. As it seen on the screen, memory has restarted in a same manner. It shows that program works successfully.

Remote Side interface:



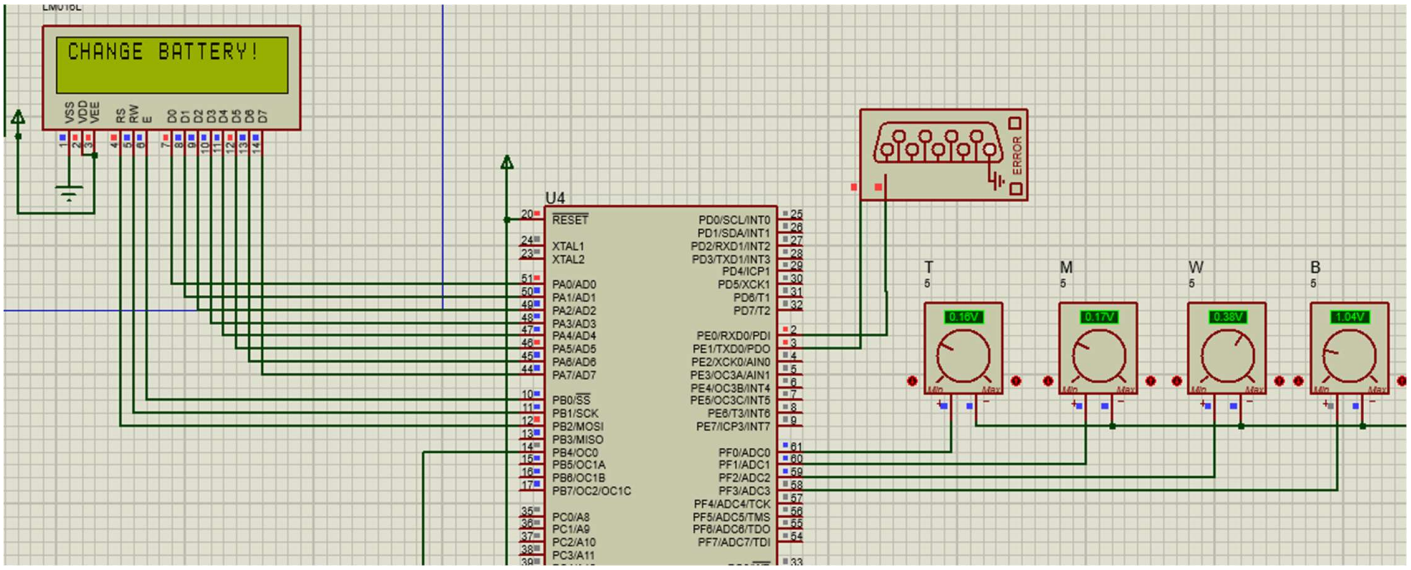
In this part of the project, there are 4 four sensors, which are represented by a adjustable voltage supplies. Also, there are DC motor and motor driver, which are shown at the bottom of the screen . In addition to these elements, 16x2 LCD is shown on the left side of the screen. Thanks to LCD screen, all parameters that sensors send to Remote MCU are displayed on. All connections are shown at the first part of the verification, clearly, and they all work properly.

“The values on the LCD and voltage supplies are changed in this part to see system is work correctly.”



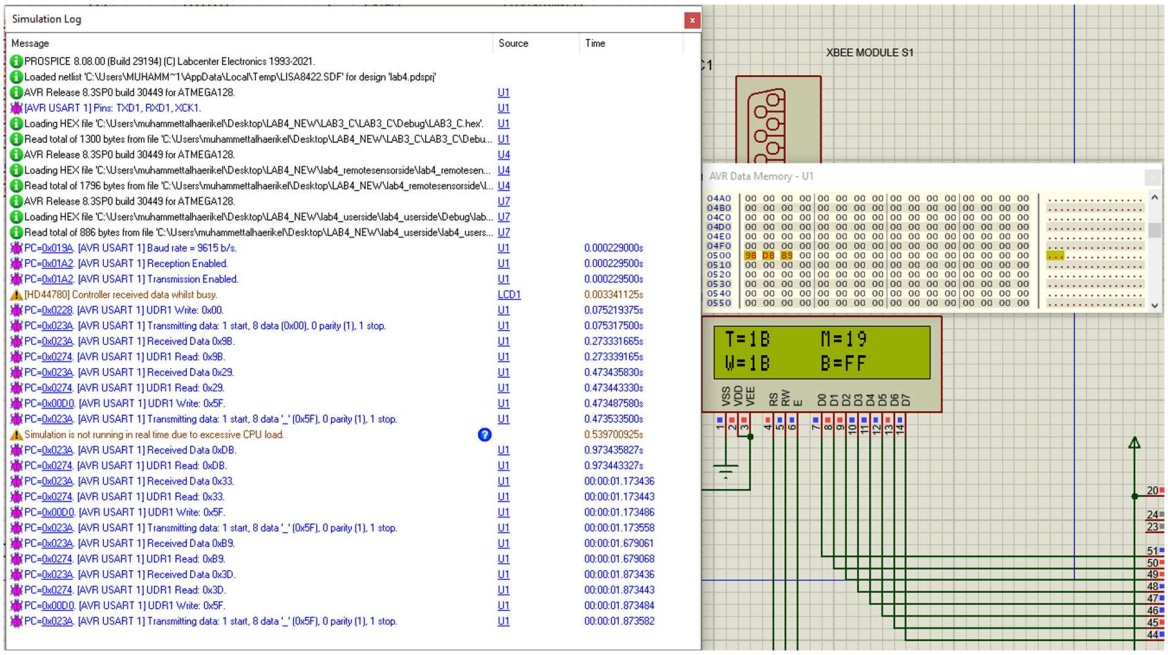
In this verification process, the values have been changed. And system has stopped again. As it is seen on the new screenshot, the new parameters are shown on the LCD, which proofs that the system work successfully.

Low Battery Scenario :



In this part, it is shown that when the battery level is lower than 3.2V , the user is informed by the system, which sends a message “Change Battery”. For this illustration, battery voltage level has been chosen 1.04V , which is lower value than 3.2V.Again, it is proven that the system is worked in a correct form.

General MCU and Remote Sensor Wireless Communication Process:

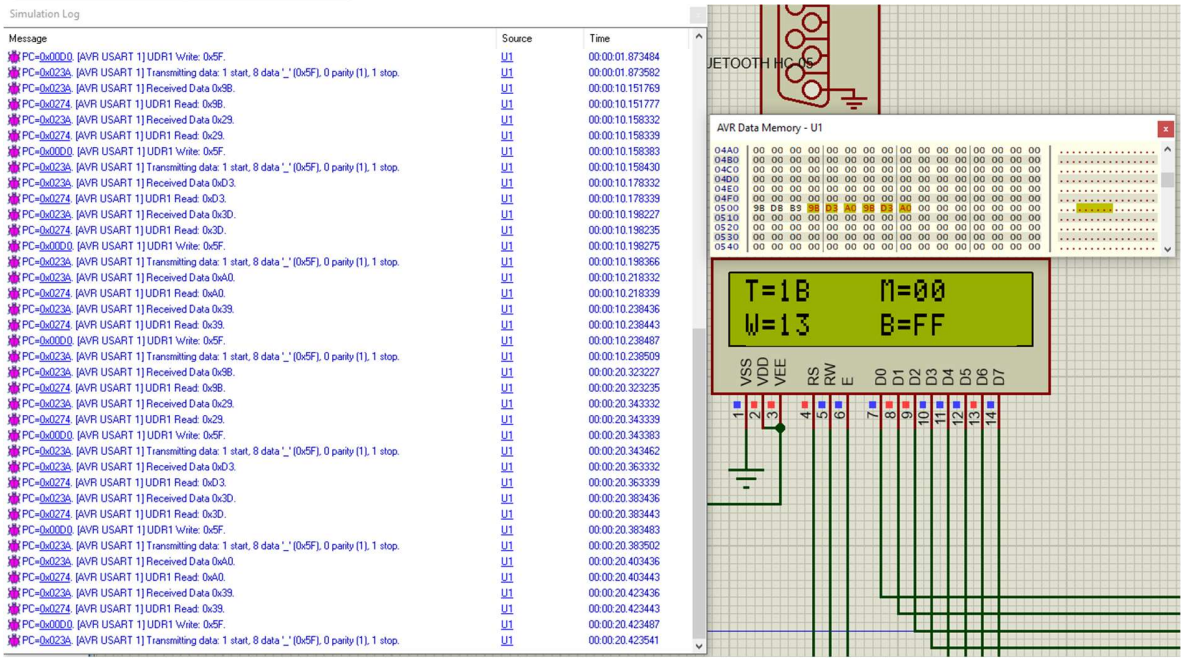


➔ The remote sensor sense three different parameter from sensor and display them on the LCD screen.After that, the remote sensors transmit the values with their parameter ID.

Data Packet [6:5]	Description
00	Temperature
01	Moisture
10	Water Level
11	Battery Level

As it seen that all values are sent with their parameter id, correctly.

➔



After 10 second of simulation, The remote sensor sense three different parameter from sensor and display them on the LCD screen. After that, the remote sensors transmit the values with their parameter ID. And the general MCU saved all transmitted values into data memory.

Code Section:

User Side Interface:

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#define F_CPU 8000000
#define BaudRate 9600
#define BR_Calc ((F_CPU/16/BaudRate)-1)   //Baud rate calculator
#include <util/delay.h>
#define KEY_PRT PORTC
#define KEY_DDR DDRC
#define KEY_PIN PINC
unsigned char keypad_value;
unsigned char ENABLE_ASCII_TO_HEX_CONVERTER;
unsigned char data;
unsigned char keypad[4][3] =
{      {'1','2','3'},
       {'4','5','6'},
       {'7','8','9'},
       {'.','0','.'}
};

unsigned char cursor_position=0x80;

int keypad_pressed()
{
unsigned char colloc, rowloc;
KEY_DDR = 0b01111000;
KEY_PRT = 0xFF;

while(1)
{
       do
       {
              KEY_PRT &= 0b00000111; // ground all rows
              colloc = (KEY_PIN & 0b00000111); // read columns
       } while (colloc != 0b00000111); // check all open
       do
       {
              do
              {
                     _delay_ms(20); // call delay
                     colloc = (KEY_PIN & 0b00000111); //any pressed?
              } while (colloc == 0b00000111); // check pressed
              _delay_ms(20); // for debounce
              colloc = (KEY_PIN & 0b00000111); // read columns
       } while (colloc == 0b00000111); //wait for pressed

       while (1)
       {
              KEY_PRT = 0b11110111; // ground row 0
              colloc = (KEY_PIN & 0b00000111); // read columns
              if (colloc != 0b00000111) // column detected
              {
                     rowloc = 0; // save row location
                     break; // exit while loop
              }
              KEY_PRT = 0b11101111; // ground row 1
              colloc = (KEY_PIN & 0b00000111); // read columns
              if (colloc != 0b00000111) // column detected
              {
                     rowloc = 1; // save row location
```

```
                        break; // exit while loop
                }
                KEY_PRT = 0b11011111; // ground row 2
                colloc = (KEY_PIN & 0b00000111); // read columns
                if (colloc != 0b00000111) // column detected
                {
                        rowloc = 2; // save row location
                        break; // exit while loop
                }
                KEY_PRT = 0b10111111; // ground row 3
                colloc = (KEY_PIN & 0b00000111); // read columns
                if (colloc != 0b00000111) // column detected
                {
                        rowloc = 3; // save row location
                        break; // exit while loop
                }
        }

        // check column; send result to PORTA
        if (colloc == 0b00000110){
                return  (keypad[rowloc][0]);
        }

        else if ( colloc == 0b00000101){
                return  (keypad[rowloc][1]);
        }

        else if (colloc == 0b00000011){
                return  (keypad[rowloc][2]);
        }
}

}
void LCD_sent_command(unsigned char value){


        PORTB &= ~(1<<1);
        PORTB &= ~(1<<2);
        PORTA = value;
        PORTB |=(1<<0);
        _delay_ms(1);
        PORTB &= ~(1<<0);
        _delay_us(100);



}

void LCD_sent_data(unsigned char value){

        PORTA = value;
        PORTB |= (1<<2);
        PORTB &= ~(1<<1);

        PORTB |= (1<<0);
        //_delay_ms(1);
        PORTB &= ~(1<<0);
        //_delay_us(100);
        cursor_position++;
        if(cursor_position==0x94){
                LCD_sent_command(0xc0);
                cursor_position=0xc0;
        }
        else if(cursor_position==0xd4){
```

```
                LCD_sent_command(0x94);
                cursor_position=0x94;
        }
        else if(cursor_position==0xa8){
                LCD_sent_command(0xd4);
                cursor_position=0xd4;
        }
        else if(cursor_position==0xe8)
        {
                LCD_sent_command(0x01);
                LCD_sent_command(0x80);
                cursor_position=0x80;
        }
}

void transmit_data(unsigned char t_data){
        while(!(UCSR0A & (1<<UDRE0)));
        UDR0 = t_data;
}




int main(void)
{
        /*///
        EIMSK = (1<<INT7);
        EICRA = (1<<ISC71)|(1<<ISC70);
        *////
        ////////////////////////// USART0 Settings ////////////////////////
        //High and low bits for baud rate
        UBRR0H = (BR_Calc >> 8);
        UBRR0L = BR_Calc;
        /////////////////
        UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);  //8 bit data format
        UCSR0B = ((1 << TXEN0) | (1 << RXEN0) | (1<< RXCIE0)| (1<< TXCIE0));
        ////////////////////////////////////////////////////////////////
        ENABLE_ASCII_TO_HEX_CONVERTER=0;
        DDRA = 0xff;
        DDRB = 0xff;
        DDRD = 0xff;
        PORTE |= (1<<7);
        LCD_sent_command(0x38);
        LCD_sent_command(0x0F);
        LCD_sent_command(0x01);
        set_sleep_mode(SLEEP_MODE_IDLE);
        sei();


        while(1){
                keypad_value=keypad_pressed();
                LCD_sent_data(keypad_value);
                _delay_us(10);
                LCD_sent_data(0x20);
                transmit_data((keypad_value));



        }


return 0;
}
```

```
ISR(USART0_RX_vect)
{

	if(ENABLE_ASCII_TO_HEX_CONVERTER==0){
		unsigned char temp2=UDR0;
		LCD_sent_data(temp2);
		_delay_us(15);
	}
	else{
	unsigned char temp = UDR0;
	unsigned char lowerbits,higherbits;
	higherbits=0x0F &(temp>>4);
	lowerbits=(0x0F & temp);
	if(higherbits>=10){
		higherbits+=7;
	}
	if(lowerbits>=10){
		lowerbits+=7;
	}
	data = higherbits + 0x30;
	LCD_sent_data(data);
	_delay_us(15);
	data = lowerbits + 0x30;
	LCD_sent_data(data);
	_delay_us(15);
	LCD_sent_data(0x20);
	}
}



ISR(USART0_TX_vect)
{
	ENABLE_ASCII_TO_HEX_CONVERTER=1;
	if(keypad_value=='3') // IF RESET REQUEST IS SENT THEN RESET USERSIDE MCU AS WELL
	{
		WDTCR = (1<<WDCE)|(1<<WDE)  ;
		WDTCR = 0;
		WDTCR = (1<<WDE) ;
	}

}
```

Remote Sensor Side:

```
/*
 * lab4_remotesensorside.c
 *
 * Created: 27.06.2021 18:23:26
 * Author : EmreDuman
 */
#include <util/crc16.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#define F_CPU 8000000
#define BaudRate 9600
#define BR_Calc ((F_CPU/16/BaudRate)-1)   //Baud rate calculator
#include <util/delay.h>
unsigned char tempature_level,moisture_level,water_level,battery_level;
unsigned char cursor_position=0x80;
unsigned char data_transmission_active = 0x00 ;
```

```
unsigned char t_data;
void LCD_sent_command(unsigned char value){


	PORTB &= ~(1<<1);
	PORTB &= ~(1<<2);
	PORTA = value;
	PORTB |=(1<<0);
	_delay_ms(1);
	PORTB &= ~(1<<0);
	_delay_ms(1);



}

void LCD_sent_data(unsigned char value){
	_delay_ms(1);
	PORTA = value;
	PORTB |= (1<<2);
	PORTB &= ~(1<<1);

	PORTB |= (1<<0);
	_delay_ms(1);
	PORTB &= ~(1<<0);
	_delay_ms(1);
	/*cursor_position++;
	if(cursor_position==0x90){
		LCD_sent_command(0xc0);
		cursor_position=0xc0;
	}
	else if(cursor_position==0xd0)
	{
		LCD_sent_command(0x01);
		LCD_sent_command(0x80);
		cursor_position=0x80;
	}
	*/
}

void transmit_data(unsigned char t_data){
	while(!(UCSR0A & (1<<UDRE0)));
	UDR0 = t_data;
}

void convert_ascii(unsigned char data){
	unsigned char temp = data;
	unsigned char lowerbits,higherbits;
	higherbits=0x0F &(temp>>4);
	lowerbits=(0x0F & temp);
	if(higherbits>=10){
		higherbits+=7;
	}
	if(lowerbits>=10){
		lowerbits+=7;
	}
	data = higherbits + 0x30;
	LCD_sent_data(data);
	_delay_us(10);
	data = lowerbits + 0x30;
	LCD_sent_data(data);
	_delay_us(10);
```

```
}

int CRC_3(unsigned char data_in){
	unsigned char generator= 0b00110101;
	generator=generator<<2;
	unsigned char shift_no=0x02;
	while(1){
		if( !(data_in & 0x80))  // check msb
		{
			if(shift_no==0)
			{
				break;
			}
			else
			{
				data_in=data_in<<1;
				shift_no--;
			}
		}
		else{
			data_in=data_in^generator;
		}

	}
	data_in=data_in>>2;
	return data_in;
}


int CRC_11(unsigned char command,unsigned char data)
{
	unsigned char generator=0b00110101;
	generator=generator<<2;
	unsigned char counter=10;


	while(1){

		if( !( data & 0x80))  // check msb
		{
			if(counter==0)
			{
				break;
			}
			else
			{
				data=( (data<<1) | (command>>7) );
				command = (command<<1);

				counter--;
			}
		}
		else{
			data=data^generator;

		}

	}
	data=data>>2;
	return data;
}

void TRANSMIT(unsigned char data_out){
	while(!(UCSR0A & (1<<UDRE0)));
```

```
	UDR0=data_out;
}

void Start_sensing()
{
	LCD_sent_command(0x01);
	LCD_sent_data('T');
	LCD_sent_data('=');
	LCD_sent_command(0x88);
	LCD_sent_data('M');
	LCD_sent_data('=');
	LCD_sent_command(0xc0);
	LCD_sent_data('W');
	LCD_sent_data('=');
	LCD_sent_command(0xc8);
	LCD_sent_data('B');
	LCD_sent_data('=');

	ADMUX = 0x60; // AVCC Vref, ADC0 single ended // data will be left-justified
	ADCSRA |= (1<<ADSC); // start conversion
	while ((ADCSRA & (1<<ADIF))==0);
	// wait for end of conversion
	ADCSRA |= (1<<ADIF); // write 1 to clear ADIF flag
	tempature_level = ADCH;
	LCD_sent_command(0x82);
	convert_ascii(tempature_level);

	ADMUX = 0x61; // AVCC Vref, ADC1 single ended // data will be left-justified
	ADCSRA |= (1<<ADSC); // start conversion
	while ((ADCSRA & (1<<ADIF))==0);
	// wait for end of conversion
	ADCSRA |= (1<<ADIF); // write 1 to clear ADIF flag
	moisture_level = ADCH;
	LCD_sent_command(0x8A);
	convert_ascii(moisture_level);

	ADMUX = 0x62; // AVCC Vref, ADC2 single ended // data will be left-justified
	ADCSRA |= (1<<ADSC); // start conversion
	while ((ADCSRA & (1<<ADIF))==0);
	// wait for end of conversion
	ADCSRA |= (1<<ADIF); // write 1 to clear ADIF flag
	water_level = ADCH;
	LCD_sent_command(0xc2);
	convert_ascii(water_level);

	ADMUX = 0x63; // AVCC Vref, ADC3 single ended // data will be left-justified
	ADCSRA |= (1<<ADSC); // start conversion
	while ((ADCSRA & (1<<ADIF))==0);
	// wait for end of conversion
	ADCSRA |= (1<<ADIF); // write 1 to clear ADIF flag
	battery_level = ADCH;
	LCD_sent_command(0xcA);
	convert_ascii(battery_level);
	if(battery_level<0b10100011){
		LCD_sent_command(0x01);
		LCD_sent_command(0x80);
		LCD_sent_data('C');
		LCD_sent_data('H');
		LCD_sent_data('A');
		LCD_sent_data('N');
		LCD_sent_data('G');
		LCD_sent_data('E');
		LCD_sent_data(0x20);
		LCD_sent_data('B');
```

```
		LCD_sent_data('A');
		LCD_sent_data('T');
		LCD_sent_data('T');
		LCD_sent_data('E');
		LCD_sent_data('R');
		LCD_sent_data('Y');
		LCD_sent_data('!');
	}
}

void START_MOTOR()
{

	TCCR0=0x61;
	if((0x00<=moisture_level) & (moisture_level<=0x07)){
		//LOW MOISTURE LEVEL , NEED FULL POWER ! %100 duty cycle
		OCR0=255;

	}
	else if((0x08<=moisture_level) & (moisture_level<=0x0F)){
		// DUTY CYCLE = %75
		OCR0=191;

	}
	else if((0x10<=moisture_level) & (moisture_level<=0x17)){
		// DUTY CYCLE = %50
		OCR0=127;

	}
	else if((0x18<=moisture_level) & (moisture_level<=0x1F)){
		// DUTY CYCLE = %25
		OCR0=63;

	}

}

void STOP_MOTOR()
{

	OCR0=0;
}

void set_timer3_5s(){
	////////////////////////// TIMER3 Settings ////////////////////////
	TCNT3 = 0;
	TCCR3A =  0x00 ;
	TCCR3B = 0x0D ;
	ETIMSK = (1<<OCIE3A);
	OCR3A = 39061;
	//////////////////////////////////////////////////////////////////
}

void set_timer1_5s(){
	////////////////////////// TIMER1 Settings ////////////////////////
	TCNT1 = 0;
	TCCR1A =  0x00 ;
	TCCR1B = 0x0D ;
	TIMSK = (1<<OCIE1A);
	OCR1A = 39061;
	//////////////////////////////////////////////////////////////////

}
```

```c
int main(void)
{
    ////////////////////// USART0 Settings ///////////////////////
    //High and low bits for baud rate
    UBRR0H = (BR_Calc >> 8);
    UBRR0L = BR_Calc;
    ////////////////
    UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);  //8 bit data format
    UCSR0B = ((1 << TXEN0) | (1 << RXEN0)| (1<< RXCIE0) );
    ////////////////////////////////////////////////////////////////
```

```c
    DDRA = 0xff;
    DDRB = 0xff;
    LCD_sent_command(0x38);
    LCD_sent_command(0x0C);
    LCD_sent_command(0x01);
```

```c
    DDRF = 0; // make Port F an input
    ADCSRA = 0x87; // enable ADC and select ck/128
```

```c
    Start_sensing();
    START_MOTOR();
    set_timer3_5s();

    _delay_ms(200);

    t_data =(tempature_level | (1<<7));
    TRANSMIT(t_data);
    _delay_ms(200);
    t_data = (0x20 | CRC_11(0x20,(tempature_level | (1<<7))));
    TRANSMIT(t_data);  /// LOG REQUEST IS SENT
    _delay_ms(500);

    t_data=((water_level| (1<<7)| (1<<6)));
    TRANSMIT(t_data);
    _delay_ms(200);
    t_data = (0x20 | CRC_11((0x20),(water_level| (1<<7)| (1<<6))));
    TRANSMIT(t_data);  /// LOG REQUEST IS SENT
    _delay_ms(500);

    t_data = ((moisture_level| (1<<7) | (1<<5)));
    TRANSMIT(t_data);
    _delay_ms(200);
    t_data=(0x20 | CRC_11(0x20,(moisture_level| (1<<7) | (1<<5))));
    TRANSMIT(t_data);  /// LOG REQUEST IS SENT
    _delay_ms(500);
```

```c
    set_sleep_mode(SLEEP_MODE_IDLE);
    sei();
```

```c
                        /// infinite loop
```

```
	while (1){

		sleep_mode();
	}



	return 0;


}
ISR(TIMER3_COMPA_vect){

	TCCR3B=0;
	STOP_MOTOR();
	set_timer1_5s();


}




ISR(TIMER1_COMPA_vect)
{
	TCCR1B=0;
	Start_sensing();
	START_MOTOR();


	if ((data_transmission_active == 0xFF)) // IF RESET REQUEST IS ARRIVED , YOU CAN START SEND THE
DATA !
	{


		t_data =(tempature_level | (1<<7));
		TRANSMIT(t_data);
		_delay_ms(20);
		t_data = (0x20 | CRC_11(0x20,(tempature_level | (1<<7))));
		TRANSMIT(t_data);  /// LOG REQUEST IS SENT
		_delay_ms(20);
		t_data=((water_level| (1<<7)| (1<<6)));

		TRANSMIT(t_data);
		_delay_ms(20);
		t_data = (0x20 | CRC_11((0x20),(water_level| (1<<7)| (1<<6))));
		TRANSMIT(t_data);  /// LOG REQUEST IS SENT
		_delay_ms(20);
		t_data = ((moisture_level| (1<<7) | (1<<5)));
		TRANSMIT(t_data);
		_delay_ms(20);
		t_data=(0x20 | CRC_11(0x20,(moisture_level| (1<<7) | (1<<5))));
		TRANSMIT(t_data);  /// LOG REQUEST IS SENT
		_delay_ms(20);

	}
	set_timer3_5s();
}


ISR(USART0_RX_vect)
{
	unsigned char received_data;
```

```
	received_data=UDR0;
	if ((CRC_3(received_data)==0))
	{
		if (((received_data | !(1<<6))==0xFF) &  ((received_data | !(1<<5))==0xFF))
		{	////REPEAT REQUEST IS ARRIVED , send all data again
			t_data =(tempature_level | (1<<7));
			TRANSMIT(t_data);
			_delay_ms(20);
			t_data = (0x20 | CRC_11(0x20,(tempature_level | (1<<7))));
			TRANSMIT(t_data);  /// LOG REQUEST IS SENT
			_delay_ms(20);
			t_data=((water_level| (1<<7)| (1<<6)));

			TRANSMIT(t_data);
			_delay_ms(20);
			t_data = (0x20 | CRC_11((0x20),(water_level| (1<<7)| (1<<6))));
			TRANSMIT(t_data);  /// LOG REQUEST IS SENT
			_delay_ms(20);
			t_data = ((moisture_level| (1<<7) | (1<<5)));
			TRANSMIT(t_data);
			_delay_ms(20);
			t_data=(0x20 | CRC_11(0x20,(moisture_level| (1<<7) | (1<<5))));
			TRANSMIT(t_data);  /// LOG REQUEST IS SENT
			_delay_ms(20);


		}

		else if(  ((received_data | !(1<<6))==0xFF)&((received_data & (1<<5))==0x00)  )
		{
			//TRANSMIT(0x31);
		}

		else if ((received_data==0))
		{ /// RESET REQEST IS ARRIVED
			data_transmission_active = 0xFF ;
		}

	}
	else
	{
		TRANSMIT(0x60|CRC_3(0x60)); /// CRC3 FAIL TRANSMIT REPEAT REQUEST
	}



}
```

General MCU side (LAB-3):

```
/*
 * LAB3_C.c
 *
 * Created: 8.06.2021 13:23:20
 * Author : EmreDuman
 */
#include <avr/wdt.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <avr/eeprom.h>
#define MST_WD_MENU "\rEnter MCU WD timer delay (& period):\rA-0.5s\rB-1s\rC-2s \r\0"
#define MST_WD_MENU2 "\rEnter Remote Sensor WD timer delay (& period):\rA-0.5s\rB-1s\rC-2s \r\0"
#define READY "MCU is initiliazed!\0"
#define USER_MENU "Enter choice: 1-Mem Dump 2-Last Entrey 3-Restart\0"
#define Reset_Request_Declaration " READY!\0"   //"READY ! - Reset Request is transmitted to RF !\r\0"
#define ACKNOWLEDGE "ACKNOWLEGDE ARRIVED !\0"
#define F_CPU 8000000
#define BaudRate 9600
#define BR_Calc ((F_CPU/16/BaudRate)-1)  //Baud rate calculator
#define UCSR0C_ADDR 0x95
#include <util/delay.h>

unsigned char  WD_EEPROM_GLO;
unsigned char *x = (unsigned char *)0x0500;  // INTMEMEND = 0x10CD
unsigned char *y;
unsigned char DATA_PACKET = 0;
unsigned char data_out;
unsigned char data_in;
unsigned char bl_data_in;
unsigned char bl_data_out;
unsigned char user_tr_buffer[128];
unsigned char user_tr_index = 0; // keeps track of character index in buffer
unsigned int temp_clock;

void TRANSMIT_RF(unsigned char data_out){
       while(!(UCSR1A & (1<<UDRE1)));
       UDR1=data_out;
}
void TRANSMIT_BL(unsigned char data_out){
       while(!(UCSR0A & (1<<UDRE0)));
       UDR0=data_out;
}

int CRC_3(unsigned char data_in){
       unsigned char generator= 0b00110101;
       generator=generator<<2;
       unsigned char shift_no=0x02;
       while(1){
              if( !(data_in & 0x80))  // check msb
              {
                     if(shift_no==0)
                     {
                            break;
                     }
                     else
                     {
                            data_in=data_in<<1;
                            shift_no--;
                     }
              }
              else{
                     data_in=data_in^generator;
              }

       }
       data_in=data_in>>2;
       return data_in;
}

void REPEAT_REQUEST(){
       data_out=0b01100000;
       data_out |= CRC_3(data_out);
       TRANSMIT_RF(data_out);
```

```
}



void UserBufferOut(unsigned char user_tr_index,unsigned char user_tr_buffer[])
{
	while(!(user_tr_buffer[user_tr_index]=='\0')){
		user_tr_index++;
	}
	unsigned char i = 0;
	while (user_tr_index > 0) {
		while(!(UCSR0A & (1<<UDRE0)));
		user_tr_index--;
		UDR0 = user_tr_buffer[i++];
	}
}



int CRC_11(unsigned char command,unsigned char data)
{
	unsigned char generator=0b00110101;
	generator=generator<<2;
	unsigned char counter=10;


	while(1){

		if( !( data & 0x80))  // check msb
		{
			if(counter==0)
			{
				break;
			}
			else
			{
				data=( (data<<1) | (command>>7) );
				command = (command<<1);

				counter--;
			}
		}
		else{
			data=data^generator;

		}

	}
	data=data>>2;
	return data;
}
void save_into_memory(unsigned char variable){
	*x=variable;
	x++;

}

int READ_EEPROM(unsigned int adress)
{
	while( EECR & (1<<EEMWE) );   /// WAİT UNTİL EEMWE İS 0

			EEAR = adress;
			EECR |= (1<<EERE);
			return EEDR;
```

```
}

void WRITE_EEPROM(unsigned int adress,unsigned char value)
{
	while( (SPMCSR & (1<< SPMEN )) );

	while( EECR & (1<<EEMWE) );   /// WAİT UNTİL EEMWE İS 0
		EEAR = adress;
		EEDR = value;

		EECR |= (1<<EEMWE);
		EECR |= (1<<EEWE);

}



int main(void){


	////////////////////////// USART0 Settings ////////////////////////
	//High and low bits for baud rate
	UBRR0H = (BR_Calc >> 8);
	UBRR0L = BR_Calc;
	/////////////////
	UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);  //8 bit data format
	UCSR0B = ((1 << TXEN0) | (1 << RXEN0) | (1<< RXCIE0)); //  | (1 << TXCIE0)
	////////////////////////// USART1 Settings ////////////////////////
	//High and low bits for baud rate
	UBRR1H = (BR_Calc >> 8);
	UBRR1L = BR_Calc;
	/////////////////
	UCSR1C = (1 << UCSZ11) | (1 << UCSZ10);  //8 bit data format
	///////////////TRANSMİT AND RECEİVE ON ////////////////////////
	UCSR1B = ((1 << TXEN1) | (1 << RXEN1) | (1<< RXCIE1));

	////////////// XMEM INITILIZATION ///////////
	MCUCR |= 0b10000000;
	XMCRB &= 0b11111000;
	XMCRA |= 0b00010010;
	XMCRA &= 0b10011111;

	/*//////////////////////         *START*        //////////////////////
	////////////////////////// WATCH DOG TIMER MENU  //////////////////////
	//////////////////////////                                    //////////////////////

	unsigned char selection;

	if (READ_EEPROM(0) & READ_EEPROM(1))
	{
		strcpy((char *)user_tr_buffer,MST_WD_MENU);
		UserBufferOut(user_tr_index,user_tr_buffer);

		while(!(UCSR0A & (1<<RXC0)));
		selection=UDR0;
		if(selection=='A')
		{
			eeprom_busy_wait();
			eeprom_write_byte((unsigned char*)0,0x00);
			eeprom_write_byte((unsigned char*)1,0b00000101);
			WDTCR = (1<<WDCE)|(1<<WDE)  ;
			WDTCR = (1<<WDP0) | (1<<WDP2);
			WDTCR = (1<<WDE) | (1<<WDP0) | (1<<WDP2) ;
```

```
	}
	if(selection=='B')
	{
		eeprom_busy_wait();
		eeprom_write_byte((unsigned char*)0,0x00);
		eeprom_write_byte((unsigned char*)1,0b00000110);
		WDTCR = (1<<WDCE)|(1<<WDE)  ;
		WDTCR = (1<<WDP2) | (1<<WDP1);
		WDTCR = (1<<WDE) | (1<<WDP2) | (1<<WDP1) ;
	}
	if(selection=='C')
	{
		eeprom_busy_wait();
		eeprom_write_byte((unsigned char*)0,0x00);
		eeprom_write_byte((unsigned char*)1,0b00000111);
		//
		WDTCR = (1<<WDCE)|(1<<WDE)  ;
		WDTCR = (1<<WDP0) | (1<<WDP2) | (1<<WDP1);  //// WDTCR = 0b00000111
		WDTCR = (1<<WDE) | (1<<WDP0) | (1<<WDP2) | (1<<WDP1);
		//
	}

}
else
{
	unsigned char WD_EEPROM = READ_EEPROM(1);
	WDTCR = (1<<WDCE)|(1<<WDE)  ;
	WDTCR =  WD_EEPROM ;
	WDTCR = (1<<WDE) | WD_EEPROM ;
}

WD_EEPROM_GLO = READ_EEPROM(1);

if ((READ_EEPROM(2) & READ_EEPROM(3))==0xFF)
{
	strcpy((char *)user_tr_buffer,MST_WD_MENU2);
	UserBufferOut(user_tr_index,user_tr_buffer);

	while(!(UCSR0A & (1<<RXC0)));
	selection=UDR0;
	if(selection=='A')
	{/// SET TİMER1 for 0.5s  -> with 256 prescale 16bit - 15625 = 0xC2F7
		eeprom_busy_wait();
		eeprom_write_byte((unsigned char*)2,0xC2);
		eeprom_write_byte((unsigned char*)3,0xF7);

		TCNT1 = 0xC2F7;
		TCCR1A=0;
		TCCR1B=4;
		TIMSK |= (1<<TOIE1);
		temp_clock=0xC2F7;

	}
	else if(selection=='B')
	{/// SET TİMER1 for 1s      -> with 256 prescale 16bit - 31250 = 0x85EE
		TCNT1 = 0x85EE;

		eeprom_busy_wait();

		eeprom_write_byte((unsigned char*)2,0x85);
		eeprom_write_byte((unsigned char*)3,0xEE);
```

```
                TCCR1A=0;
                TCCR1B=4;
                TIMSK |= (1<<TOIE1);
                temp_clock=0x85EE;
        }
        else if(selection=='C')
        {/// SET TİMER1 for 2s      -> with 256 prescale 62500 = 0x=0BDC
                eeprom_busy_wait();
                while(!(eeprom_is_ready()));
                eeprom_write_byte((unsigned char*)2,0x0B);
                eeprom_write_byte((unsigned char*)3,0xDC);

                TCNT1 = 0x0BDC;
                TCCR1A=0;
                TCCR1B=4;
                TIMSK |= (1<<TOIE1);
                temp_clock=0x0BDC;
        }
}
else
{
        TCNT1H = READ_EEPROM(2);
        TCNT1L = READ_EEPROM (3);
        TCCR1A=0;
        TCCR1B=4;
        TIMSK |= (1<<TOIE1);
        temp_clock=READ_EEPROM(2);
        temp_clock =(temp_clock<<7);
        temp_clock |=READ_EEPROM(3);
}

//////////////////////////                        //////////////////////////
////////////////////////// WATCH DOG TIMER MENU  //////////////////////////
//////////////////////////          *END*                //////////////////////////*/



strcpy((char *)user_tr_buffer,READY);
UserBufferOut(user_tr_index,user_tr_buffer);

////// Reset Request is displayed to the USER SCREEN//////
strcpy((char *)user_tr_buffer,Reset_Request_Declaration);
UserBufferOut(user_tr_index,user_tr_buffer);



//////////////USER MENU OPTİON////////////////////
strcpy((char *)user_tr_buffer,USER_MENU);
UserBufferOut(user_tr_index,user_tr_buffer);


sei();
////// Transmit the Reset Request ///////
while(!(UCSR1A & (1<<UDRE1))); /// Wait the flags
UDR1 = 0 | CRC_3(0);

set_sleep_mode(SLEEP_MODE_IDLE);
while(1)
{
 sleep_mode();
}
```

```
	}

	ISR(USART1_RX_vect)
{

	    /******** TIMER ***********************
	       ////// WATCHDOG TİMER is disabled ////
	       WDTCR = (1<<WDCE)|(1<<WDE)  ;
	       WDTCR = WD_EEPROM_GLO;
	       ******** TIMER1 is stopped !
	       TCCR1B=0;
	       ********* TIMER ***********************/

	       data_in = UDR1;

	       if(!(data_in & (1<<7)))
	       {
	              ************ COMMAND PACKET İN ***************
	              if( !(DATA_PACKET & (1<<7)) )
	              {
	                     ///// TOS HAS NO DATA PACKET //////
	                     if(!(CRC_3(data_in)))
	                     { ********* CRC3 TRUE ********

	                            if(  (data_in | !(1<<6)) &  !(data_in & (1<<5)) )
	                            {
	                                   ********* ACKNOWLEDGE PACKET IS ARRIVED **************
	                                   strcpy((char *)user_tr_buffer,ACKNOWLEDGE);
	                                   UserBufferOut(user_tr_index,user_tr_buffer);

	                            }
	                            else
	                            {
	                                   if(  (data_in | !(1<<6)) &  (data_in | !(1<<5)) )
	                                   { ****************** REPEAT REQUEST IS ARRIVED
****************

	                                          TRANSMIT_RF(DATA_PACKET);  /// Transmit the data where
top of the stack

	                                   }
	                            }


	                     }
	                     else
	                     {
	                            ********* CRC3 FALSE ********
	                            ******* TRANSMİT REPEAT ****
	                            REPEAT_REQUEST();
	                     }
	              }
	              else
	              {
	                     if( !(CRC_11(data_in,DATA_PACKET)) )
	                     { ********** CRC_CHECK_11 PASS **********
	                            if( (data_in | !(1<<5)) & !(data_in & (1<<6)) )
	                            { ******* LOG REQUEST IS ARRIVED **********
	                                   save_into_memory(DATA_PACKET);
	                                   data_out = 0b01000000;
	                                   data_out |= CRC_3(data_out);
	                                   TRANSMIT_RF(data_out);
```

```
                                }


                        }
                        else
                        { ////////// CRC_CHECK_11 FAİL ///////////
                                REPEAT_REQUEST();
                        }
                }

        }



        else
        {
                ////////////// DATA PACKET İN ///////////////
                DATA_PACKET=data_in;
        }

    /*//////// TIMER //////////////////////
    TCNT1=temp_clock;           // Re-initilized the timer

    TCCR1B=4;
    //// WATCHDOG TİMER is reinitialized ////
    WDTCR = (1<<WDCE)|(1<<WDE)  ;
    WDTCR = 0b00001111;
    ///////// TIMER //////////////////////*/



}

ISR(USART0_RX_vect)
{

    /*//////// TIMER //////////////////////
    ////// WATCHDOG TİMER is disabled ////
    WDTCR = (1<<WDCE)|(1<<WDE)  ;
    WDTCR = WD_EEPROM_GLO;
    //////// TIMER1 is stopped !
    TCCR1B=0;
    ///////// TIMER //////////////////////*/

    bl_data_in = UDR0;

                if (bl_data_in==0x31)
                {
                        y=x;
                        while(!(x==(unsigned char*)0x04FF))
                        {

                                x=x-1;        // SAVE INITIAL ADRESSES OF MEMORY
                                TRANSMIT_BL(*x);
                                //TRANSMIT_BL(0x20);


                        }
                        x=y; //// RETURN INITAL ADRESSES OF MEMORY

                }
                else if(bl_data_in==0x32)
                {
                        y=x;          // SAVE INITIAL ADRESSES OF MEMORY
```

```
                        x=x-1;
                        TRANSMIT_BL(*x);
                        //TRANSMIT_BL(0x20);

                        x=y;
                        //TRANSMIT_BL('\r');
                }
                else if(bl_data_in==0x33)
                {
                        //RESTART FUNCTİON WİTH WATCHDOG
                        WDTCR = (1<<WDCE)|(1<<WDE)  ;
                        WDTCR = 0;
                        WDTCR = (1<<WDE) ;
                        //TRANSMIT_BL('\r');
                }



/*//////// TIMER //////////////////////
TCNT1=temp_clock;           // Re-initilized the timer
TCCR1B=4;
//// WATCHDOG TİMER is reinitialized ////
WDTCR = (1<<WDCE)|(1<<WDE)  ;
WDTCR = 0b00001111;
///////// TIMER //////////////////////*/



}

        ISR(TIMER1_OVF_vect)
        {

                ////// Reset Request is displayed to the USER SCREEN//////
                strcpy((char *)user_tr_buffer,Reset_Request_Declaration);
                UserBufferOut(user_tr_index,user_tr_buffer);

                ////// Transmit the Reset Request ///////
                while(!(UCSR1A & (1<<UDRE1))); /// Wait the flags
                UDR1 = 0 | CRC_3(0);

        }


        /*ISR(BADISR_vect)
        {
         //DEBUG THE UNHANDLED İNTERRUPT
        }
        */
```

Conclusion:

In this experiment , It has been learned how to overcome when it is faced any situations or solve a problem. The project has several challenges such as displaying hex values to ASCII, keypad configuration, etc. We have learned how to fix and simulate the errors of our multiple MCU configuration. In this project, divide and conquer method is followed, and this method was very helpful in order to create a large scale of project. Later, the AVR code is designed and debugged for each line. Then, in Proteus 8 , the circuit board is created and components are placed as designed, and it is learned that the Proteus 8 is good for the real life simulation.