FeSCADA & Arduino

Introduction

An automation application was developed to show the possibilities of FeSCADA. By the end of this paper the reader will learn how FeSCADA can work with Arduino.

1. Description
2. Hardware
3. Sensors and actuators
4. Arduino program
5. FeMODBUS communication setup
6. FeSCADA project
7. Database setup
8. Web server development
9. Conclusions

1) Description

In the following pages an application is described for monitoring the temperature in the home and outside, detect the presence of peoples in 2 different areas, and to be able to start/stop: a buzzer, a LED, a room light and a fan. The outputs can be switched on/off either from the local computer or remotely, from a web browser (i.e. a mobile phone with internet access).

2) Hardware

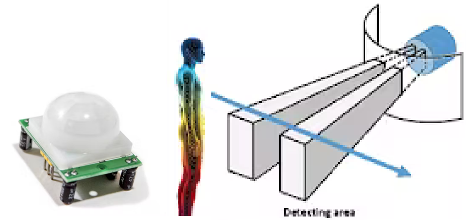
The hardware is composed of one Arduino Mega board with an Ethernet shield and an IO shield. We used an USB cable to connect and download the program and to supply the voltage for hardware.

Arduino Mega board Ethernet shield IO shield

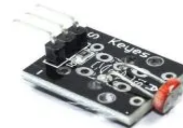
3) Sensors and actuators

Inputs

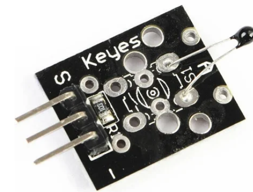
Two Passive Infra-Red (PIR) sensors. They are giving On/Off signals if a person is moving in the scanning area. The sensors sensitivity range between 6 to 7 meters (20 feet) and the detection angle is 110 degrees x 70 degrees.



One photoresistor sensor which is detecting light.

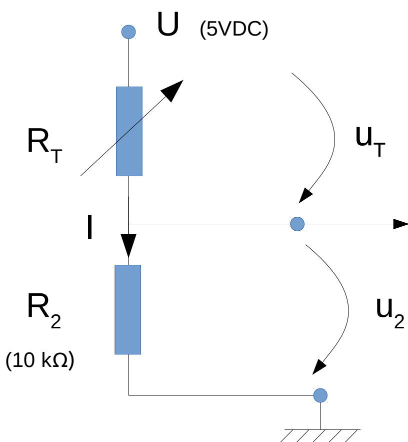


One thermistor to measure the ambient temperature.



Thermistor formula: $\frac{1}{T} = a + b \cdot \ln R_T + c \cdot (\ln R_T)^3$

where: a = $1.4 \cdot 10^{-3,}$, b = $2.37 \cdot 10^{-4}$, c = $9.9 \cdot 10^{-8}$.



R_T – is the Thermistor resistance

R_2 – is the reference resistance (10kΩ)

u_2 – is the voltage measured by Arduino with 10 bits precision. 0...5VDC will be converted to a number between 0...1024.

$U = u_T + u_2 = I \cdot (R_T + R_2) \Rightarrow I = U / (R_T + R_2)$

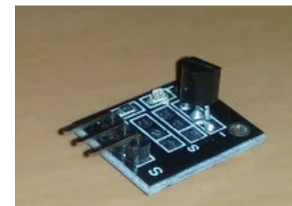
$u_T = R_T \cdot I = U - u_2 \Rightarrow R_T \cdot U / (R_T + R_2) = U - u_2$

$R_T = R_2 \cdot (U / u_2 - 1)$

$R_T = 10000 \cdot (1024 / u_2 - 1)$

One LM35 temperature sensor with the characteristics:

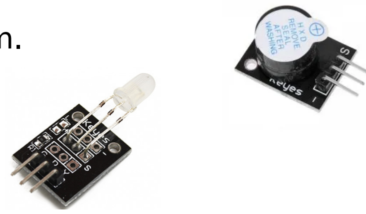
- temperature range: -55 ºC to 155 ºC
- output scale = 10 mV/ºC
- output at 25ºC = 250 mV



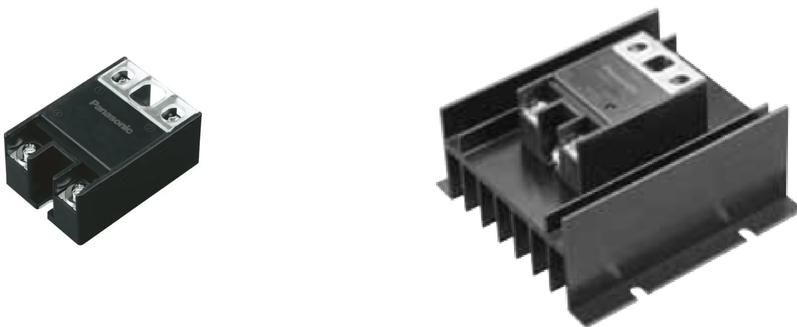
Outputs

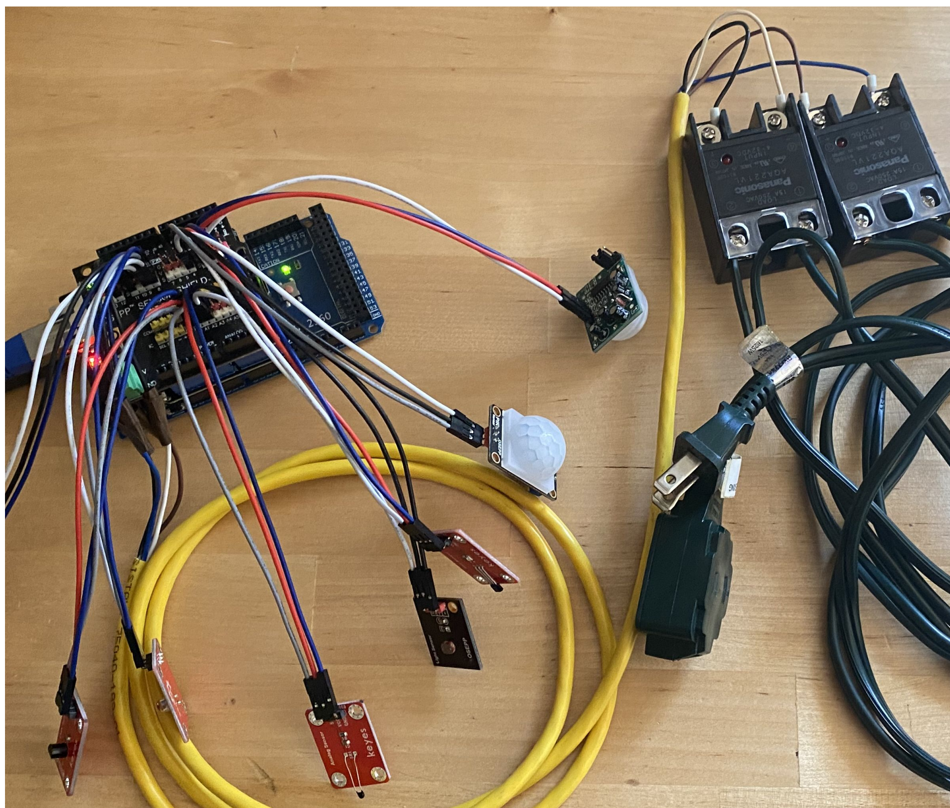
One Buzzer module to send an audible alarm.

One LED module to signal a visual warning.



Two solid state relays (SSR) with which we can switch on or off an electric load of up to 15A, at 75 – 240VAC. The relay is commanded with 5VDC.





Arduino wiring pins.

Sensor/Actuator	PIN	Description
PIR1 - digital input	2	Presence sensor – zone 1
Hall magnetic - digital input	3	Magnetic field sensor
PIR2 - digital input	4	Presence sensor – zone 2
LED - digital output	8	Light signal
Buzzer - digital output	9	Sound signal
SSR 1 - digital output	11	Solid Sate Relay – Room light
SSR 2 - digital output	12	Solid State Relay – Fan start
LM35 temperature – analog input	A0	Temperature sensor
Thermistor – analog input	A1	Temperature sensor
Photoresistor – analog input	A2	Light sensor

Hardware prices.

Name	Unit price	Qty	Price	Description
Arduino Mega	$40	1	$40	Main board - microcontroller
Ethernet shield	$25	1	$25	Ethernet TCP/IP
IO extension shield	$10	1	$10	IOs Connection board
PIR detector	$5	2	$10	Digital sensor - presence
Photoresistor	$4	1	$4	Analog sensor - light
Thermistor	$1	1	$1	Analog sensor - temperature
LM35 temperature	$2	1	$2	Analog sensor - temperature
Buzzer	$2	1	$2	Digital output - sound
LED	$0.5	1	$0.5	Digital output - light
Solid State Relay Panasonic AQA221VL	$23	2	$46	Digital output - relay
	TOTAL =		**$140.5**	

If the SSR load is bigger than 5A then a heat sink is recommended. One can mount the SSR on a metal plate or one can buy a heat sink.

Panasonic AQP-HS-J10A Standard heat sink (15A) - $25/unit x 2 = $50.

4) Arduino program

The Modbus TCP/IP protocol is used to communicate with Arduino. Modbus is a data communication protocol originally published by Modicon in 1979 for use with its PLCs. Modbus has become a de facto standard communication protocol and is now a commonly available means of connecting industrial electronic devices. Modbus is popular in industrial environments because it is openly published and royalty-free. It was developed for industrial applications, is relatively easy to deploy and maintain compared to other standards, and places few restrictions on the format of the data to be transmitted.

Useful links:
1) https://en.wikipedia.org/wiki/Modbus
2) https://www.arduino.cc/en/ArduinoModbus/ArduinoModbus

Arduino Program

```
#include <SPI.h>
#include <Ethernet.h>
#include <ArduinoModbus.h>

int i, j, iVal, iReg[10];
float fVal;
uint8_t values[10];

// Enter a MAC address and IP address for your controller below.
byte mac[] = {  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

// The IP address will be dependent on your local network
IPAddress ip(192, 168, 1, 17);

// Initialize the Ethernet server library with the IP address and port you want to use
EthernetServer server(502);
ModbusTCPServer modbusTCPServer;
EthernetClient client;

void setup() {
   //initialize inputs:
  pinMode(2, INPUT);   pinMode(3, INPUT);  pinMode(4, INPUT);
  pinMode(5, INPUT);   pinMode(6, INPUT);  pinMode(7, INPUT);
  //initialize outputs:
  pinMode(8, OUTPUT);    pinMode(9, OUTPUT);    pinMode(10, OUTPUT);
  pinMode(11, OUTPUT);  pinMode(12, OUTPUT);  pinMode(13, OUTPUT);
```

```
  Serial.begin(9600); // Open serial communications and wait for port to open
  Ethernet.begin(mac, ip); // Start the Ethernet connection and the server
  if (Ethernet.hardwareStatus() == EthernetNoHardware) {
     // Check for Ethernet hardware present
    Serial.println("Ethernet shield was not found.  Sorry, can't run without hardware. :(");
    while (true) {
      delay(1); // do nothing, no point running without Ethernet hardware
    }
  }

  if (Ethernet.linkStatus() == LinkOFF) {
    Serial.println("Ethernet cable is not connected.");
  }
  server.begin(); // start the server
  Serial.print("server is at ");   Serial.println(Ethernet.localIP());

  if (!modbusTCPServer.begin()) {    // start the Modbus TCP server
    Serial.println("Failed to start Modbus TCP Server!");
    while (1);
  }

  //Configure 20 Modbus coils, inputs and holding registers starting at address 0x00
  modbusTCPServer.configureDiscreteInputs(0x00, 20);
  modbusTCPServer.configureCoils(0x00, 20);
  modbusTCPServer.configureHoldingRegisters(0x00, 20);
}

void loop() {
  EthernetClient client = server.available();  // listen for incoming clients
  if(client) {
    Serial.println("new client");  // a new client connected
    modbusTCPServer.accept(client);  // let the Modbus TCP accept the connection

    while (client.connected()) {
      // poll for Modbus TCP requests, while client connected
      delay(10); modbusTCPServer.poll();

      iReg[0]++; //counter to check connection
      if(iReg[0]>999) iReg[0]=0;  modbusTCPServer.holdingRegisterWrite(0, iReg[0]);

      //Analog Inputs
      iVal = analogRead(A0);  //discard the first reading
      iVal = analogRead(A0); fVal=(3*fVal+iVal)/4;   //software low pass filter
      iReg[1] = (int) fVal;         modbusTCPServer.holdingRegisterWrite(1, iReg[1]);
      iReg[2] = analogRead(A1);     modbusTCPServer.holdingRegisterWrite(2, iReg[2]);
      iReg[3] = analogRead(A2);     modbusTCPServer.holdingRegisterWrite(3, iReg[3]);
      iReg[4] = analogRead(A3);     modbusTCPServer.holdingRegisterWrite(4, iReg[4]);
      iReg[5] = analogRead(A4);     modbusTCPServer.holdingRegisterWrite(5, iReg[5]);
      iReg[6] = analogRead(A5);     modbusTCPServer.holdingRegisterWrite(6, iReg[6]);

      //Analog outputs
      iReg[7] = modbusTCPServer.holdingRegisterRead(7);  analogWrite(44, iReg[7]);
      iReg[8] = modbusTCPServer.holdingRegisterRead(8);  analogWrite(45, iReg[8]);
```

```
      //Digital Inputs and Outputs
      updateIO();
    }
    Serial.println("client disconnected");
  }
}

void updateIO() {
 for(i=1;i<7;i++) {
   j=1+i;
   modbusTCPServer.discreteInputWrite(i, digitalRead(j)); // read the inputs
   values[i] = modbusTCPServer.coilRead(i);  // read the current value of the coil
   j=7+i;
   if (values[i]) digitalWrite(j, HIGH); else digitalWrite(j, LOW); // write the outputs
 }
}
```

5) FeMODBUS communication setup

The FeMODBUS software is a Modbus client. It will connect to Arduino, which is a Modbus server to read inputs and to write outputs.

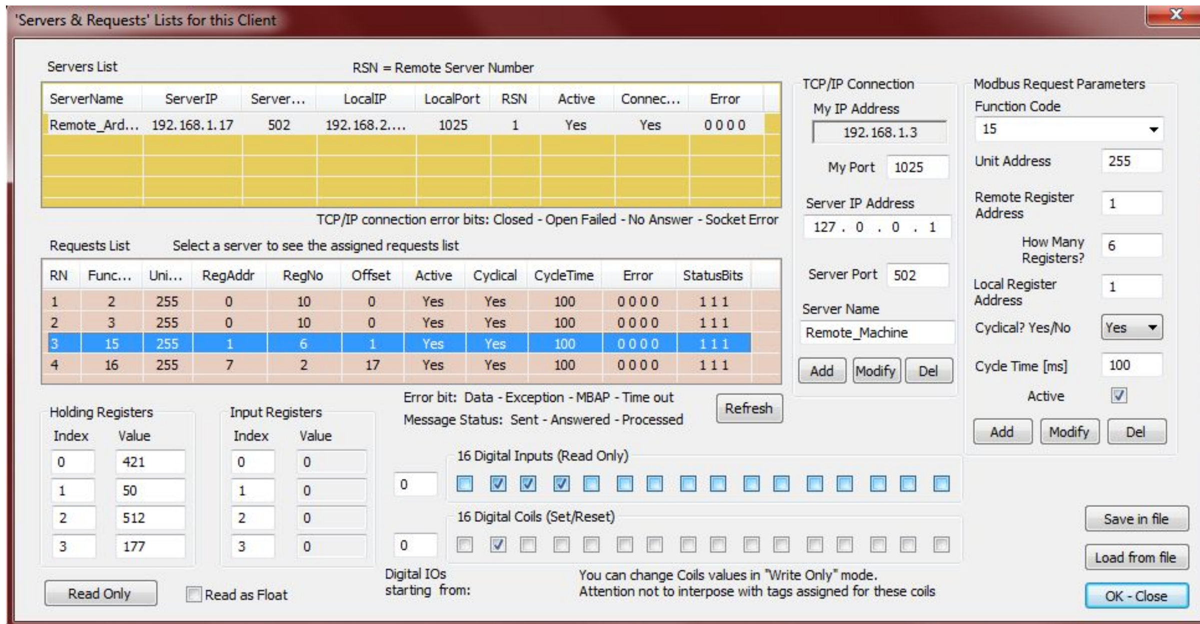
In the picture below FeMODBUS was setup to connect at the address 192.168.1.17 and to cyclically (every 100ms) send 4 requests:

- Function 2 – read 10 digital inputs from address 0 and copy them locally in digital inputs area, from address 0.

- Function 3 – read 10 holding registers from address 0 and copy them locally in holding registers area, from address 0.

- Function 15 – write 6 coils (digital outputs) from local coils area, address 1, to remote coils area, starting from address 1.

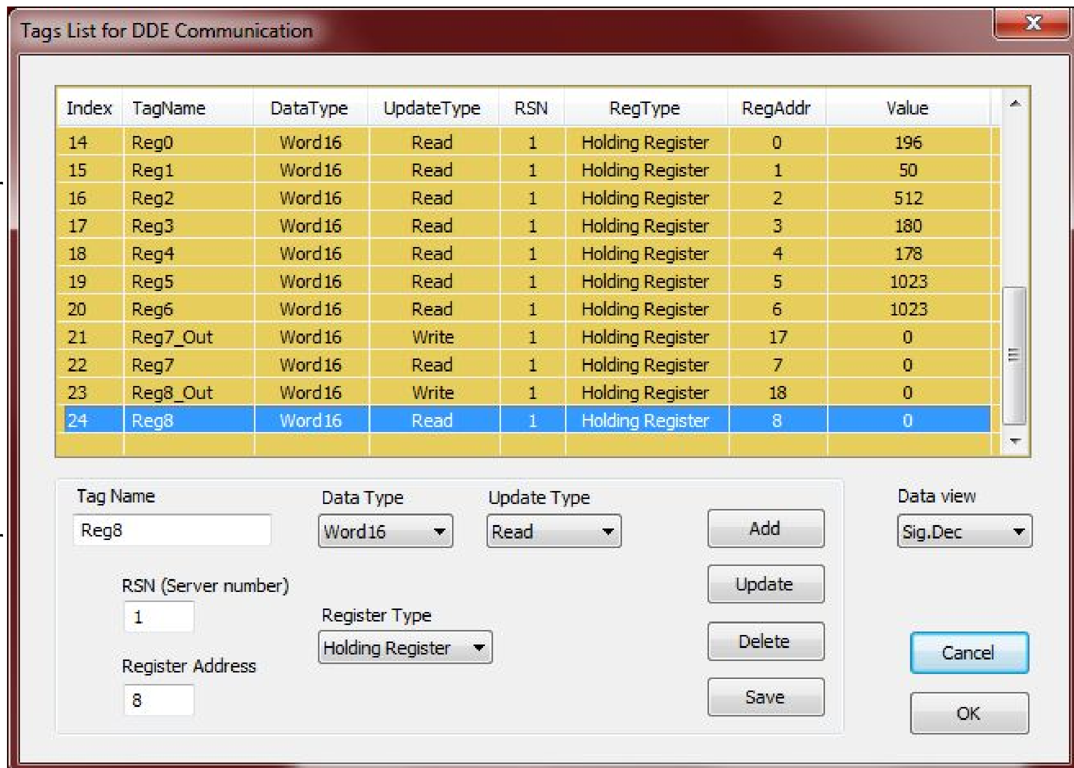
- Function 16 – write 2 holding registers to remote addresses 7 and 8 from local holding registers address 17 and 18.



Using the local registers of FeMODBUS, we define tags for the DDE communication with FeSCADA. In the picture below one can see a snapshot of the Tags List dialog window.

The tag name “Reg8” is assigned to read the holding register number 8 from the Remote Server Number (RSN) 1.

The tag name “Reg7_Out” is assigned to write the holding register number 17 for the same RSN 1.



6) FeSCADA project

The first step in a FeSCADA project is to define the DDE communication channels and the tags. In the picture below one can see that we defined one DDE channel as channel number 1: DDE_Application = "MB" and DDE_Topic = "TAGS".

Every tag has an internal name used in FeSCADA and a DDE Name for communication with other DDE servers.

Tags Setup

Tags List

No	Tag Name	DDE Name	DDE...	Data Type	Update T...	Value
32	Input0	Inp0	1	Integer	Read	0
33	Input1	Inp1	1	Integer	Read	0
34	Input2	Inp2	1	Integer	Read	1
35	Input3	Inp3	1	Integer	Read	1
36	Input4	Inp4	1	Integer	Read	0
37	Input5	Inp5	1	Integer	Read	0
38	Output1	Out1	1	Integer	Read/Write	1
39	Output2	Out2	1	Integer	Read/Write	0
40	Output3	Out3	1	Integer	Read/Write	0
41	Output4	Out4	1	Integer	Read/Write	1
42	Output5	Out5	1	Integer	Read/Write	0
43	Counter	Reg0	1	Integer	Read/Write	835
44	Temp_LM35	Reg1	1	Integer	Read	23.437500

DDE Channels

No	DDE Ap...	DDE Topic	Con...
1	MB	TAGS	Yes
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			

Tag Name: Output1 | Data Type: Integer | Update Type: Read/Write | Add

DDE Name: Out1 | Update

Initial Value: 0 | Delete

DDE Channel: 1 | Search

Scaled | Max Eng Value: 0 | Offset Value: 0 | Max Raw Value: 0

SCALED_VALUE = Max_Eng_Value x (RAW_VALUE + Offset_Value) / Max_Raw_Value

DDE Application

DDE Topic

Update | Delete

Save All | Close

From FeMODBUS the program will:

- read 6 inputs;
- read/write 5 outputs;
- read 6 registers;
- read/write 2 registers.

Tags List

No	Tag Name	DDE Name	DDE...	Data Type	Update T...	Value
38	Output1	Out1	1	Integer	Read/Write	1
39	Output2	Out2	1	Integer	Read/Write	0
40	Output3	Out3	1	Integer	Read/Write	0
41	Output4	Out4	1	Integer	Read/Write	1
42	Output5	Out5	1	Integer	Read/Write	0
43	Counter	Reg0	1	Integer	Read/Write	839
44	Temp_LM35	Reg1	1	Integer	Read	23.437500
45	Thermistor	Reg2	1	Integer	Read	490
46	Light_sensor	Reg3	1	Integer	Read/Write	2.050781
47	Reg6_In	Reg7	1	Integer	Read	0.000000
48	Reg6_Out	Reg7_Out	1	Integer	Read/Write	0
49	Reg7_In	Reg8	1	Integer	Read	0
50	Reg7_Out	Reg8_Out	1	Integer	Read/Write	0

The tag “Temp_LM35”, reading the “Reg1” register from FeMODBUS, which is the analog input A0 from Arduino, was scaled to show the value in Celsius degrees. For this we set:

MaxEngValue = 500 *OffsetValue= 0* *MaxRawValue = 1024*

This is because 5VDC is 1024 and for LM35 we have 250mV at 25°C with 10mV/°C.

The “Light_sensor” tag was scaled more simple, to show 100% when the input is 1024. *MaxEngValue = 100* *OffsetValue= 0* *MaxRawValue = 1024*

For the “Thermistor” tag we needed to write a logic program that will compute the real temperature based on the measurement of the thermistor resistance. The program is based on the thermistor formula presented at the beginning of this paper.

Logic Setup

Logic List

No	Name	Type	Cyclic	Trigger M...
0	Logic1	Timer	Yes	Different
1	Thermist...	Timer	Yes	Different
2	Buzzer	Timer	Yes	Equal
3	Remote	Timer	No	Equal
5	PID	Timer	Yes	Equal

Add Update Delete

Program edit

```
Log_R = LOG(10000.0 * ((1024.0/Thermistor-1)));
Temp = 1 / (0.001129148 + 0.000234125*Log_R +
         (0.0000000876741*Log_R$3));

Temp = Temp - 273.15;

Temp_F_Therm = 32 + 9.0 / 5.0 * Temp;


Temp_F_LM35 =  32 + 9.0 / 5.0 * Temp_LM35;
```

Save Reload Execute

Name: Thermistor formula

Program status: Loaded: 5, Prepared: 5, Executed: 5

Logic Type: Timer Timer Interval [sec]: 10.00 Cyclic Logic?: Yes

Time/Date — Execute Logic every: Hour; Hour: 00; Minute: 19; Day of the week: Sunday; Month of the year: January; Day of the month: 1

Logic Trigger TagName: << Del

Trigger Mode (when Tag ..... Value): Different

Trigger Value: 1 Bit No: -1

Application Tags List

No	Tag Name
1	Bit1
2	Bit3
3	c1
4	c2
5	Person_Counting_1
6	Person_Counting_2
7	Person_Reset
8	PID1_CMD
9	PID1_Out1
10	PID1_Out2
11	PID1_SP
12	PID1_STATUS
13	PID2_CMD
14	PID2_Out1
15	PID2_Out2

Cancel OK

The “Logic Trigger TagName” should be clear and “Logic Type” = Timer. In this way the program is executed 10 times per second.

Also in this program, we converted the temperatures from Celsius degrees to Fahrenheit degrees for both sensors: thermistor and LM35. For this we used memory tags: “Log_R”, “Temp”, “Temp_F_Therm” and “Temp_F_LM35”. The memory tags do not communicate with any DDE server. They are used as variables inside FeSCADA project.

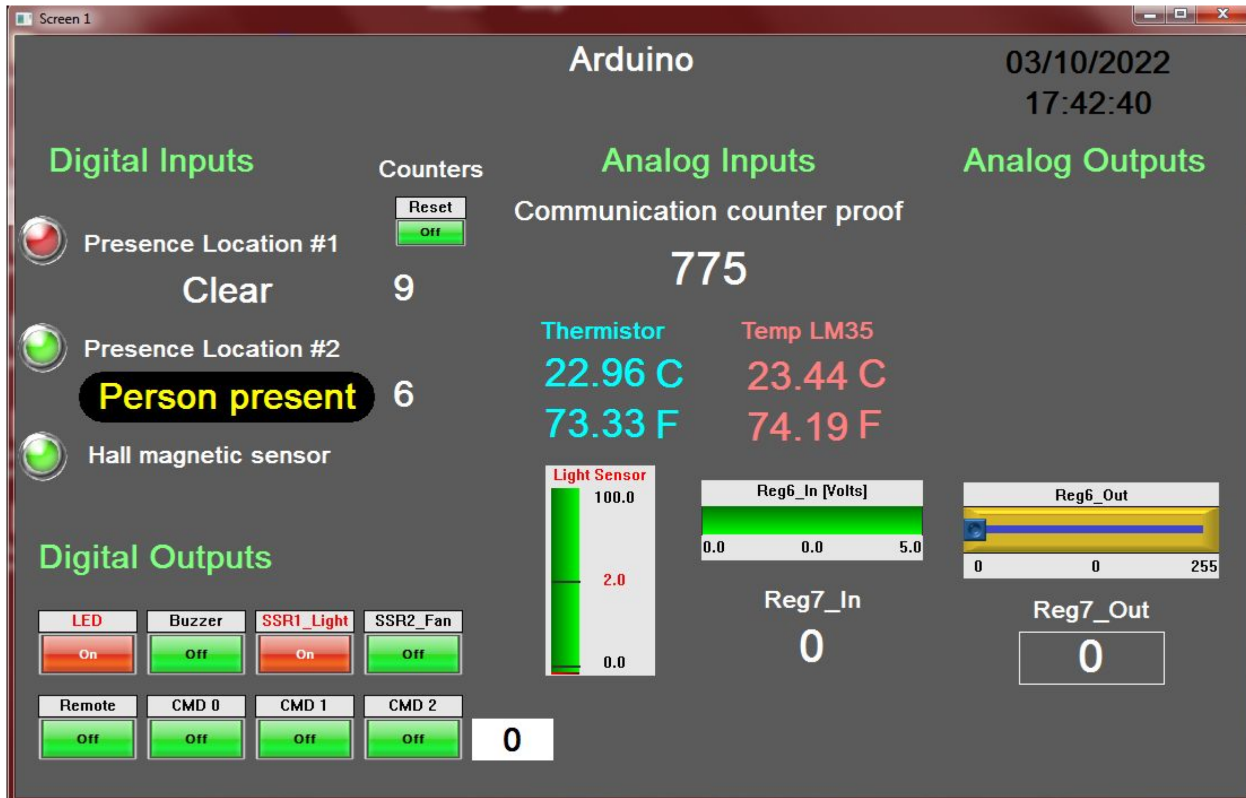
For the PIR sensors, which are read in FeSCADA as “Input1” and “Input3” tags we have written a logic program with which we count how many times the sensors were triggered.

The tag “Output1” is the LED from Arduino. It will turn on when either PIR sensor is active.

```
if Input1 OR Input3
then Output1 = 1
else Output1 = 0;

if Input1 AND NOT Bit1
{ Person_Counting_1 = Person_Counting_1 + 1;
  Bit1 = 1;
} if NOT Input1 AND Bit1 then Bit1 = 0;

if Input3 AND NOT Bit3
{ Person_Counting_2 = Person_Counting_2 + 1;
  Bit3 = 1;
} if NOT Input3 AND Bit3 then Bit3 = 0;

if Person_Reset == 1
{ Person_Counting_1 = 0;
  Person_Counting_2 = 0;
}
```

“Person_Counting_1” and “Person_Counting_2” are memory tags in which these counters are kept. This program is run also 10 times per second. The tags “Bit1” and “Bit3” are used to count only once when a sensor becomes active, at the raising edge of the signal.

Finally, we have built a panel (window) to show the data, and to put some indicators and buttons. In the picture below one can see a snapshot of this panel.



The “Communication counter proof” is the register 0 from Arduino. If this number is changing all the time from zero to 1000 and starting from zero again, this is a good proof that the communication with Arduino is okay.

For the PIR sensors we used two text messages to show “Clear” when 0 (Off) and “Person present” when 1 (On). Also 2 digital indicators are used which are changing their color: red when 0 (Off) and green when 1 (On).

The temperatures are shown with 4 numeric displays, with 2 digit precision, in both Celsius and Fahrenheit degrees.

The Light_sensor is shown with an analog indicator.

The “Reg6_Out” and “Reg7_Out” are values that can be changed by the user with the aid of a slider or by double clicking the rectangle and dialing in a new number.

“Reg6_In” and”Reg7_In” are echoed values coming in from Arduino as different registers, for confirmation.

In the lower left corner there are the buttons for commands to Arduino. The commands can turn On or Off the: LED, buzzer, room light or fan.

The commands can run in 2 ways. In a *local* mode or in a *remote* mode. If the “remote” tag is 0 (off) the user can control the top buttons locally. If the “remote” tag is 1 (on) then a logic program will copy the bit values from a command tag (“remote_cmd”) to each button. See the program in the picture below.

Also, when the remote tag is On, the tags “Reg6_Out” and “Reg7_Out” are overwritten with the values from the tags “remote_cmd2” and ”remote_val2”.

This program is executed 10 times per second.

```
if remote == 1
{
  Output2 = remote_cmd@0;
  Output4 = remote_cmd@1;
  Output5 = remote_cmd@2;

  if remote_cmd2>=0 AND remote_cmd2<=255
  then Reg6_Out = remote_cmd2;

  if remote_val2>=0 AND remote_val2<=255
  then Reg7_Out = remote_val2;

}
```

When the remote control is selected, the remote values are read from a database.

```
mysql> show tables;
+--------------------+
| Tables_in_storage |
+--------------------+
| Alarms_Active      |
| Alarms_History     |
| Command            |
| Temperatures       |
+--------------------+
4 rows in set (0.00 sec)

mysql> describe Command;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| C1    | int(11) | YES  |     | NULL    |       |
| C2    | int(11) | YES  |     | NULL    |       |
| V1    | double  | YES  |     | NULL    |       |
| V2    | double  | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
4 rows in set (0.00 sec)

mysql> describe Temperatures;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| temp1 | double  | YES  |     | NULL    |       |
| temp2 | double  | YES  |     | NULL    |       |
| temp3 | double  | YES  |     | NULL    |       |
| c1    | int(11) | YES  |     | NULL    |       |
| c2    | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

7) Database setup

We installed MySQL on an Ubuntu web server. We defined the database "storage" and we created in it 2 tables:

- Command

- Temperatures

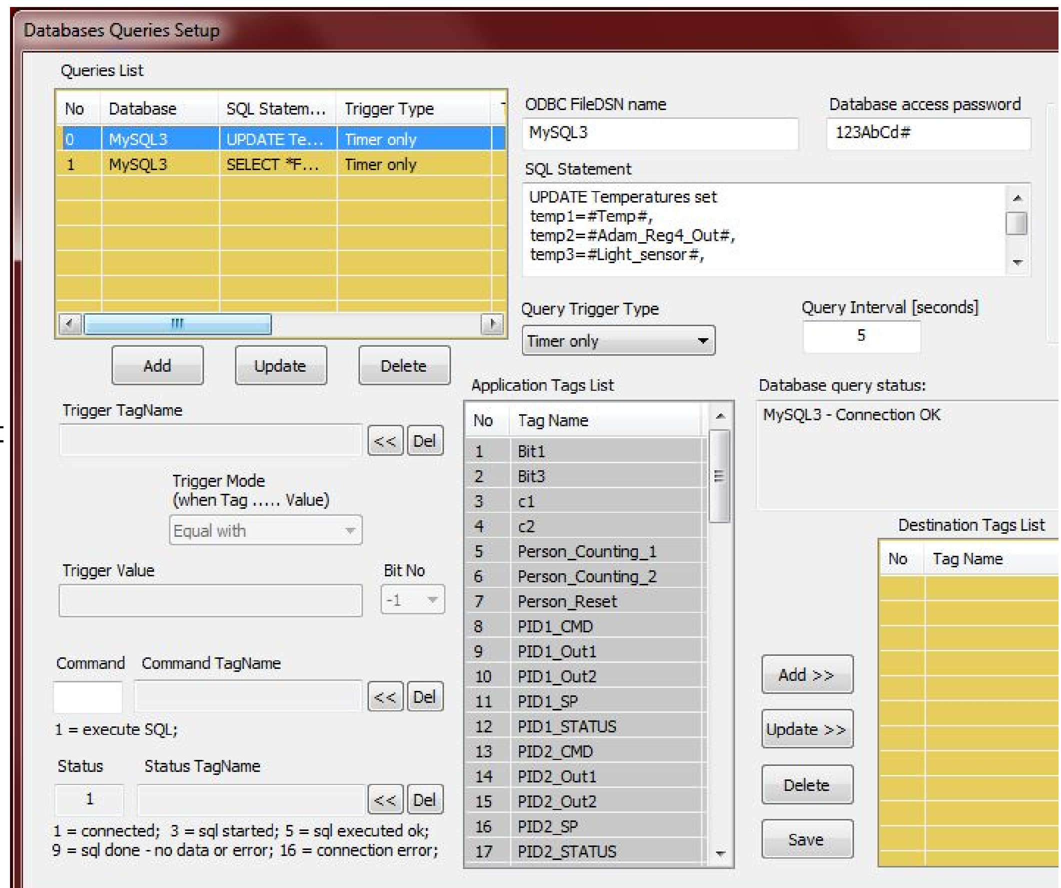
The "Command" table has 2 integers and 2 real numbers. One set of data that is updated or read.

The "Temperatures" table has 3 real numbers and 2 integers. One set of values that is updated or read.

After that we installed the MySQL drivers for ODBC access from our Windows computer to the remote MySQL database.

In FeSCADA we defined two database queries. Each one will be executed every 5 seconds.

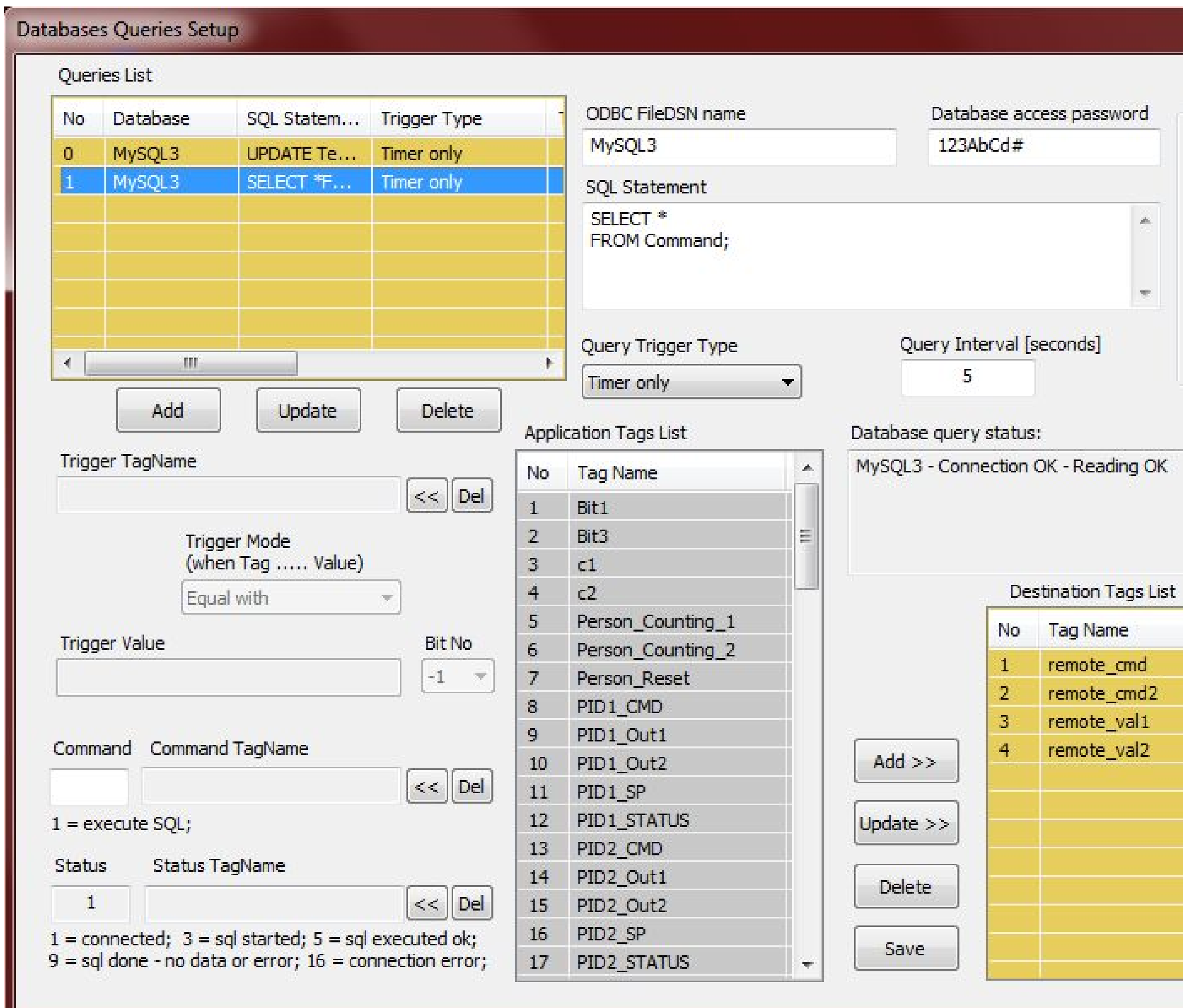
The first will update, in the table "Temperatures", the values from our temperature and light sensors and the counters from the PIR sensors.



The SQL command is like this:

UPDATE Temperatures SET
temp1=#tag1#, temp2=#tag2#, temp3=#tag3#,
c1=#tag4#, c2=#tag5# ;

The second database query will read, every 5 seconds, from the table “Command”, the values: C1, C2, V1, V2, and will copy them in the tags: “remote_cmd”, “remote_cmd2”, “remote_val1” and “remote_val2”.



In this way the remote control of the Arduino can be done by writing different values for “remote_cmd” in the MySQL database.

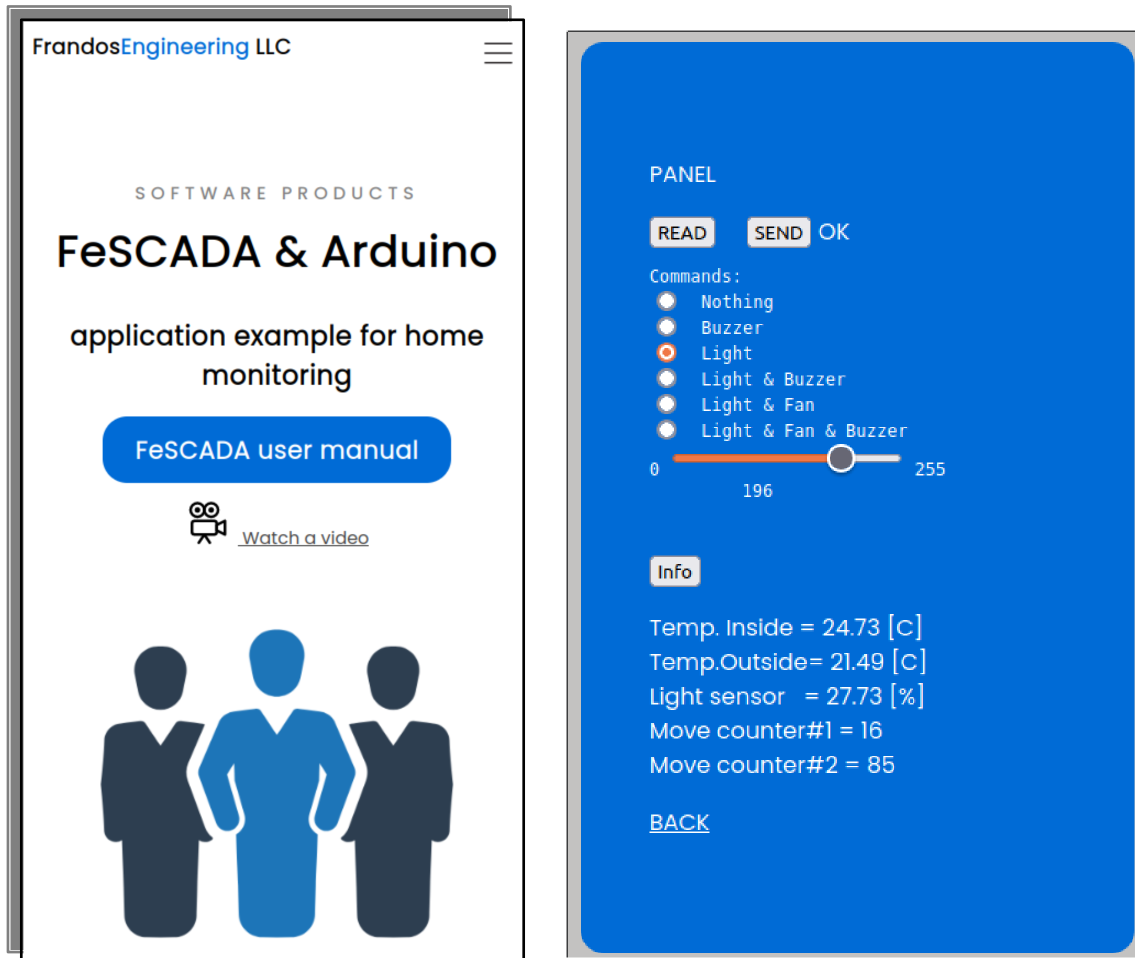
8) Web server development

On the Ubuntu server, apart of Apache web server and MySQL database, we installed PHP support. We developed a web page with HTML, CSS, JavaScript and PHP. The server is home based and the home router is setup to forward the http requests to this server.

The following web address is free dynamic DNS (from https://www.noip.com) and is based on the IP address that our Internet provider has assigned to us.

http://frandos.ddns.net/index.php

In this web page we created a panel section with radio buttons selector, a sliding button and 3 push buttons.



The top of the panel will change the value of command word C1 and the slider will alter the value of C2. The button SEND will send an HttpRequest to update the database. The button READ will send an HttpRequest to read the values from the database and update the option and the slider values.

The bottom of the panel will show the data from the table Temperatures. The information will be refreshed every 5 seconds.

In the following pages there are presented some snippets from the HTML and JavaScript program, executed in the browser, and from the PHP programs, executed on the server side.

HTML page

```
……………………..
<li id="Panel">
<blockquote>
<p> PANEL</p>
<button type="button" onclick="loadDoc('Command2.php', myFunction5)">READ </button>
   
<button type="button" onclick="sendDoc('Command3.php', myFunction6)">SEND </button>
<label readonly id="text1"> </label>

<form name='Commands' oninput="x_comm1.value = parseInt(comm1.value); y_comm2.value =
parseInt(comm2.value); document.getElementById('text1').innerHTML = '';">
<pre>
Commands:
<input type='radio' size='3' id="comm1" name='comm1' value=0> Nothing
<input type='radio' size='3' id="comm1" name='comm1' value=1> Buzzer
<input type='radio' size='3' id="comm1" name='comm1' value=2> Light
<input type='radio' size='3' id="comm1" name='comm1' value=3> Light & Buzzer
<input type='radio' size='3' id="comm1" name='comm1' value=6> Light & Fan
<input type='radio' size='3' id="comm1" name='comm1' value=7> Light & Fan & Buzzer
0 <input type="range" id="comm2" name="comm2" min="0" max="255"> 255
<output name="y_comm2" for="comm2"></output> <input type="hidden" id=x_comm1 name="x_comm1"
value=0>
</pre>
</form>

<br>
<button type="button" onclick="loadDoc('Values.txt', myFunction3)">Info </button>
<p readonly id="value"> <br> </p>

<a href="#" style="color:white">BACK</a>
</blockquote>
</li>

………………………………..
<script>
loadDoc('Command2.php', myFunction5);
var myVar = setInterval(myTimer, 5000);
```

```
function myTimer(){
loadDoc('Temperature.php', myFunction4);
}

function loadDoc(url, cFunction) {
var xhttp;
xhttp=new XMLHttpRequest();
xhttp.onreadystatechange = function() {
if (this.readyState == 4 && this.status == 200) {
cFunction(this);
}
};
xhttp.open("GET", url+"?t="+Math.random(), true);
xhttp.send();
}

function sendDoc(url, cFunction) {
var xhttp;
xhttp=new XMLHttpRequest();
xhttp.onreadystatechange = function() {
if (this.readyState == 4 && this.status == 200) {
cFunction(this);
}
};
xhttp.open("POST", url, true);

var x = document.getElementById("x_comm1");
var y = document.getElementById("comm2");
var text1 = "comm1="+x.value+"&comm2="+y.value;
xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xhttp.setRequestHeader("Content-length", text1.length);
xhttp.setRequestHeader("Connection", "close");
xhttp.send(text1);
}

function myFunction3(xhttp) {
document.getElementById("value").innerHTML = xhttp.responseText;
}

function myFunction4(xhttp) {
var myVals = JSON.parse(xhttp.responseText);
document.getElementById("value").innerHTML =
"Temp. Inside = " + myVals.temp1 + " [C] " +
"<br>Temp.Outside= " + myVals.temp2 + " [C] " +
"<br>Light sensor   = " + myVals.temp3 + " [%]" +
"<br>Move counter#1 = " + myVals.c1 +
```

```
"<br>Move counter#2 = " + myVals.c2 ;
}

function myFunction5(xhttp) {
var myArduino = JSON.parse(xhttp.responseText);
document.forms["Commands"]["comm1"].value = myArduino.cmd1;
document.forms["Commands"]["comm2"].value = myArduino.cmd2;
document.forms["Commands"]["x_comm1"].value = myArduino.cmd1;
document.forms["Commands"]["y_comm2"].value = myArduino.cmd2;
document.getElementById("text1").innerHTML = "";
}

function myFunction6(xhttp) {
document.getElementById("text1").innerHTML = xhttp.responseText;
}
</script>
</body></html>
```

PHP program – Temperatures.php   (Executed every 5 seconds)

```
<?php
require_once 'login.php';

$mysqli = new mysqli($db_hostname, $db_username, $db_password, $db_database);
if($mysqli->connect_error) {
exit('Could not connect');
}
$temp1 = -1.0;
$temp2 = -1.0;
$temp3 = -1.0;
$c1 = -1;
$c2 = -1;

$sql = "SELECT * FROM Temperatures";
$stmt = $mysqli->prepare($sql);
$stmt->execute();
$stmt->store_result();
$stmt->bind_result($temp1, $temp2, $temp3, $c1, $c2);
$stmt->fetch();
$stmt->close();

printf("{\"temp1\": \"%.02f\", \"temp2\": \"%.02f\", \"temp3\": \"%.02f\", \"c1\": \"%u\", \"c2\": \"%u\" }",
$temp1, $temp2, $temp3, $c1, $c2);
?>
```

PHP program – Command2.php (READ button)

```
<?php
require_once 'login.php';
$mysqli = new mysqli($db_hostname, $db_username, $db_password, $db_database);
if($mysqli->connect_error) {
exit('Could not connect');
}
$sql = "SELECT * FROM Command";
$stmt = $mysqli->prepare($sql);
$stmt->execute();
$stmt->store_result();
$stmt->bind_result($C1, $C2, $V1, $V2);
$stmt->fetch();
$stmt->close();
printf("{\"cmd1\": \"%u\", \"cmd2\": \"%u\", \"val1\": \"%.02f\", \"val2\": \"%.02f\" }",
$C1, $C2, $V1, $V2);
?>
```

PHP program – Command3.php (SEND button)

```
<?php
require_once 'login.php';
$mysqli = new mysqli($db_hostname, $db_username, $db_password, $db_database);
if($mysqli->connect_error) {
exit('Could not connect');
}
if ( isset($_POST['comm1']) && isset($_POST['comm2']) ) {
$c1 = get_post($mysqli, 'comm1');
$c2 = get_post($mysqli, 'comm2');
$sql = "UPDATE Command SET C1='$c1', C2='$c2' ";
$stmt = $mysqli->prepare($sql);
$result = $stmt->execute();
  if(!$result)   echo "Update command failed:" . mysqli_error() . "<br>,<br>";
  else  {
  $stmt->close();
  echo "OK";
  }
}
else  echo "No wright parameters";

function get_post($db_server, $var) {
return mysqli_real_escape_string($db_server, $_POST[$var]);
}
?>
```

9) Conclusions

An application of FeSCADA software with an Arduino microcontroller was presented.

The Arduino has sensors and actuators connected to its input and output pins.

Ethernet communication is used to exchange Modbus TCP messages between Arduino and FeMODBUS software. FeSCADA is exchanging DDE messages with FeMODBUS.

The data from the sensors is processed with logic programs in FeSCADA to compute mathematical formulas (like the thermistor formula) and other logic or arithmetic algorithms.

Different kinds of actuators: LED, buzzer, room light, fan, are controlled with push buttons from FeSCADA.

The control can be switched between local and remote. The remote control is possible by using a database.

An AMP (Apache – MySQL – PHP) server was setup and a web page was created to allow the reading and writing to a database.

With this web page, and by selecting remote access in MySCADA project, it is possible to read the home temperature, the light level, the presence sensors counters, and to control home appliances with a mobile phone connected to the Internet.