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 \mathbf{R}_2 – is the reference resistance (10k Ω) \mathbf{u}_2 - is the voltage measured by Arduino with 10 bits precision. 0...5VDC will be converted to a number between 0...1024. $U = u_T + u_2 = I \cdot (R_T + R_2) \implies I = U / (R_T + R_2)$ $u_T = R_T \cdot I = U - u_2 => R_T \cdot U / (R_T + R_2) = U - u_2$ $R_T = R_2 \cdot (U / u_2 - 1)$ $R_T = 10000 \cdot (1024 / u_2 - 1)$

One LM35 temperature sensor with the characteristics:

- temperature range: -55 °C to 155 °C

- output scale = $10 \text{ mV/}^{\circ}\text{C}$
- output at $25^{\circ}C = 250 \text{ mV}$









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	\$23	2	\$46	Digital output - relay
Panasonic AQA221VL				
	тс	TAL =	\$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) - $25/unit \times 2 = 50$.

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);
}
{ 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]);
                                 modbusTCPServer.holdingRegisterWrite(3, iReg[3]):
   iReg[3] = analogRead(A2);
   iReg[4] = analogRead(A3);
                                 modbusTCPServer.holdingRegisterWrite(4, iReg[4]);
                                 modbusTCPServer.holdingRegisterWrite(5, iReg[5]);
   iReg[5] = analogRead(A4);
   iReg[6] = analogRead(A5);
                                 modbusTCPServer.holdingRegisterWrite(6, iReg[6]);
   //Analog outputs
   iReg[7] = modbusTCPServer.holdingRegisterRead(7); analogWrite(44, iReg[7]);
   Reg[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.

JEIVI	ers List				RSN = R	emote Ser	rver Numb	er				
Serv	erName	Serv	erIP	Server	LocalIP	LocalP	ort RSN	Active	Connec.	Error	TCP/IP Connection	Modbus Request Parameters Function Code
Rem	ote_Ard	192.16	8.1.17	502	192.168.2	102	5 1	Yes	Yes	0000	192.168.1.3	15 -
											My Port 1025	Unit Address 255
			1								Server IP Address	Remote Register
					TCP/IP connec	tion error	bits: Close	d - Open Fail	ed - No Ans	wer - Socket Erro	or 127.0.0.1	Address
Requ	iests List	Sel	ect a serv	ver to see th	ne assigned rec	uests list					_	How Many 6 Registers?
RN	Func	Uni	RegAdd	dr RegN	o Offset	Active	Cyclical	CycleTime	Error	StatusBits	Server Port 502	Local Register 1
1	2	255	0	10	0	Yes	Yes	100	0000	111	Server Name	Address
2	3	255	0	10	0	Yes	Yes	100	0000	111	Remote Machine	Cyclical? Yes/No Yes 🔻
3	15	255	1	6	17	Yes	Yes	100	0000	111		Curle Tree [mail 100
7	10	233		2	17	163	ies	100	0000	111	Add Modify Del	Cycle time [ms] 100
Hold	na Reaiste	rs	Inc	ut Register	s	Error bit:	Data - Exc	ception - MBA	P - Time ou	t Refrest	1	Active 🔽
Inde	ex Valu	ie	In	dex Val	ue	Message	Status: Se	nt - Answere	d - Process	ed		Add Modify Del
0	42	L	0	0		1	6 Digital Ir	puts (Read C	(nly)			
<u> </u>	50		1	0	0							
1	50	,	-	0		-	6 Digital C	oils (Set/Rese	t)			Coursin 6
1	51.	·	2	0								save in fi
1										the second se		

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.

Index	TagName	DataType	UpdateType	RSN	RegType	RegAddr	Value	-
14	Reg0	Word16	Read	1	Holding Register	0	196	
15	Reg1	Word16	Read	1	Holding Register	1	50	
16	Reg2	Word16	Read	1	Holding Register	2	512	
17	Reg3	Word 16	Read	1	Holding Register	3	180	
18	Reg4	Word16	Read	1	Holding Register	4	178	
19	Reg5	Word 16	Read	1	Holding Register	5	1023	
20	Reg6	Word 16	Read	1	Holding Register	6	1023	
21	Reg7_Out	Word16	Write	1	Holding Register	17	0	
22	Reg7	Word16	Read	1	Holding Register	7	0	-
23	Reg8_Out	Word16	Write	1	Holding Register	18	0	
24	Reg8	Word16	Read	1	Holding Register	8	0	-
Tag N	lame	Data	Type I	Jpdate T	ype		Data viev	v
Reg	3	Word	116 🔻	Read		Add	Sig.Dec	•
	RSN (Server nu	imber)				Update		
	1	Regis	ter Type					
		L La Lak	- Desistant -	11		Delete		

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".

x Tags Setup DDE Channels Tags List No Tag Name DDE Name DDE... Data Type Update T... Value No DDE Ap... DDE Topic Con... . 0 TAGS 32 Inp0 MB Input0 1 Integer Read Yes 33 Input1 Inp1 1 Integer Read 0 34 Input2 Inp2 1 Integer Read 1 3 35 Input3 Inp3 1 Integer Read 1 36 Read 0 Input4 Inp4 1 Integer 37 Input5 Inp5 Integer Read 0 Ξ Out2 0 8 39 Output2 1 Integer Read/Write 40 Output3 Out3 Integer Read/Write 0 1 41 Output4 Out4 Integer Read/Write 1 10 42 Output5 Out5 Integer Read/Write 0 43 Integer Read/Write 835 Counter Rea0 12 44 Temp_LM35 23.437500 13 Reg1 Integer Read Ш . Tag Name Data Type Update Type **DDE** Application Add Output1 Integer -Read/Write -DDE Topic DDE Name Update Out1 Delete Initial Value DDE Channel 0 Update Delete 1 Search Max Eng Value Offset Value Max Raw Value 0 Scaled 0 0 SCALED_VALUE = Max_Eng_Value x (RAW_VALUE + Offset_Value) / Max_Raw_Value Save All Close

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

Tags List

From FeMODBUS the program will:

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

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	-
		m					

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.

	List				Program edit			
No	Name	Type (Cyclic	Trigger M	Log R = LOG(10000.0 * (()))	1024.	0/Thermistor-1)));	
0	Logic1	Timer	Yes	Different	T		000004105#Las D .	
1	Thermist	Timer	Yes	Different	(0.001129148)	+ 0.0 741≑L	og_R\$3));	
2	Buzzer	Timer	Yes	Equal				
3	Remote	Timer	No	Equal	Temp = Temp - 273.15;			
5	PID	Timer	Yes	Equal	$Temp_F_Therm = 32 + 9.0$	/ 5.0	* Temp;	
•				F	Temp_F_LM35 = 32 + 9.0 ,	/ 5.0	<pre>* Temp_LM35;</pre>	
Ac	dd Updat	Delete				Appl	ication Tags List	۲
					Save Reload Execute	No	Tag Name	
Name	e		_	Program statu	IS	NO		-6
The	ermistor formula	a		Loaded: 5, P	repared: 5, Executed: 5	1	Bit1	
						2	BITS	-
Logi	ic Type	Timer Interval I	[sec]	Cyclic Logic?		3	-2	
rog.		10.00	occ]	V		-	Derson Counting 1	-
-	er 🔹	10.00		res		6	Person_Counting_1	
Time						7	Person Reset	
Time								
Time				-	Logic Trigger TagName	8	PID1 CMD	
Time	/Date				Logic Trigger TagName	8	PID1_CMD PID1_Out1	-
Time	/Date Execute Lo	ogic every			Logic Trigger TagName	8 9 10	PID1_CMD PID1_Out1 PID1_Out2	
Time	/Date Execute Lo Hour	ogic every			Logic Trigger TagName	8 9 10 11	PID1_CMD PID1_Out1 PID1_Out2 PID1_SP	
Time	/Date Execute Lo Hour	ogic every			Logic Trigger TagName	8 9 10 11 12	PID1_CMD PID1_Out1 PID1_Out2 PID1_SP PID1_STATUS	
Time/ Fime/	/Date Execute Lo Hour Ir Minute	Day of the wee	•k	[Logic Trigger TagName	8 9 10 11 12 13	PID1_CMD PID1_Out1 PID1_Out2 PID1_SP PID1_STATUS PID2_CMD	
Time Fime Hou	/Date Execute Lo Hour In Minute	Day of the wee Sunday	łk ▼	(Trigger	Logic Trigger TagName	8 9 10 11 12 13 14	PID1_CMD PID1_Out1 PID1_Out2 PID1_SP PID1_STATUS PID2_CMD PID2_Out1	
Time/ Fime/ Hou 00 Mon	/Date Execute Lo Hour In Minute	Day of the wee Sunday Day of the mon	łk ▼	[Trigger 1	Logic Trigger TagName	8 9 10 11 12 13 14 15	PID1_CMD PID1_Out1 PID1_Out2 PID1_SP PID1_STATUS PID2_CMD PID2_Out1 PID2_Out2	

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.

	List				Program edit	
No	Name	Туре	Cyclic	Trigger M	if Input1 OR Input3	
0	Logic1	Timer	Yes	Different	else Output1 = 0;	
1	Thermist	Timer	Yes	Different	if Input1 AND NOT Bit1	
2	Buzzer	Timer	Yes	Equal	{ Person_Counting_1 = Person_Counting_1 + 1	;
3	Remote	Timer	No	Equal	Bit1 = 1; 3 if NOT Input1 AND Bit1 then $Bit1 = 0$.	
5	PID	Timer	Yes	Equal	j in nor input and brit then brit = 0,	E
					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;	
•	III			P.	}	-
12					-	1.00
1					€	
A	dd Updat	e Delete			Application Tags List	+
Ad	dd Updat	Delete			Save Reload Execute Application Tags List	•
Ad Nam	dd Updat	e Delete		Program stat	Save Reload Execute Application Tags List No Tag Name	•
Ad Nam Log	dd Updat e jic1	Delete		Program stat	Save Reload Execute Application Tags List No Tag Name Prepared: 12, Executed: 6	•
Ad Nam Log	dd Updat e jic1	e Delete		Program stat Loaded: 12,	Application Tags List Save Reload Execute tus No Tag Name 1 Bit1 2 Bit3	•
Ad Nam Log	dd Updat e gic1	Delete		Program stat Loaded: 12,	Application Tags List Save Reload Execute tus No Tag Name 1 Bit 1 2 Bit 3 3 c1	•
Ad Nam Log Log	dd Updat e gic1 ic Type	Timer Interval	[sec]	Program stat Loaded: 12, Cyclic Logic	Application Tags List Save Reload Execute Application Tags List No Tag Name 1 Bit1 2 Bit3 3 c1 4 c2	•
Ad Nam Log Log Tim	dd Updat e gic1 ic Type er 👻	Timer Interval	[sec]	Program stat Loaded: 12, Cyclic Logic Yes	 Save Reload Execute Application Tags List No Tag Name Bit1 Bit3 c? C? Second Compared Co	_1
Ad Nam Log Log Tim	dd Updat ie jic1 ic Type er 🔻	Timer Interval	l [sec]	Program stat Loaded: 12, Cyclic Logic Yes	 Save Reload Execute Application Tags List No Tag Name 1 Bit1 2 Bit3 3 c1 4 c2 5 Person_Counting 6 Person_Counting 	_1 _2
Ad Nam Log Log Tim	dd Updat ie jic1 ic Type er v	Timer Interval	[sec]	Program stat Loaded: 12, Cyclic Logic Yes	 Save Reload Execute Application Tags List No Tag Name Bit1 Bit3 C1 C2 Logic Trigger TagName Application Tags List No Tag Name Bit1 Bit3 C1 C2 Person_Counting Person_Reset 	_1 _2
Ad Nam Log Time	dd Updat e jic1 icType er •	Timer Interval	l [sec]	Program stat Loaded: 12, Cyclic Logic Yes	 Save Reload Execute Save Reload Execute Application Tags List No Tag Name 1 Bit1 2 Bit3 3 c1 4 c2 5 Person_Counting 6 Person_Counting 7 Person_Reset 8 PID1_CMD 	_1 _2
Ad Nam Log Tim	dd Updat ne nic Type er • /Date	Timer Interval	[sec]	Program stat Loaded: 12, Cyclic Logic Yes	 Save Reload Execute Save Reload Execute Application Tags List No Tag Name 1 Bit1 2 Bit3 3 c1 4 c2 5 Person_Counting 6 Person_Counting 7 Person_Reset 8 PID1_CMD 9 PID1_Out1 	_1 _2
Ad Nam Log Time,	dd Updat ne nic Type er • /Date Execute Lo	Timer Interval	l [sec]	Program stat Loaded: 12, Cyclic Logic Yes	 ✓ Save Reload Execute Application Tags List No Tag Name 1 Bit1 2 Bit3 3 c1 4 c2 5 Person_Counting 6 Person_Counting 7 Person_Reset 8 PID1_CMD 9 PID1_Out1 10 PID1_Out2) _1 _2
Ac Nam Log Tim	dd Updat ie jic 1 ic Type er • /Date Execute Lo Hour	Timer Interval 10.00	l [sec]	Program stat Loaded: 12, Cyclic Logic Yes	 Save Reload Execute Save Reload Execute Application Tags List No Tag Name 1 Bit1 2 Bit3 3 c1 4 c2 5 Person_Counting. 6 Person_Counting. 7 Person_Reset 8 PID1_CMD 9 PID1_Out1 10 PID1_Out2 11 PID1_SP) _1 _2
Ac Nam Log Time, Time,	dd Updat ie jic1 ic Type er v /Date Execute Lo Hour ir Minute	Timer Interval 10.00	I [sec]	Program stat Loaded: 12, Cyclic Logic Yes	✓ Save Reload Execute Application Tags List tus No Tag Name 1 Bit1 2 Bit3 2 Bit3 3 c1 c? ✓ 4 c2 5 Logic Trigger TagName <	_1 _2
Ac Nam Log Tim Time, Hou	dd Updat ie jic1 ic Type er v /Date Execute Lo Hour ir Minute v 19 v	Timer Interval 10.00	I [sec]	Program stat Loaded: 12, Cyclic Logic Yes	 Save Reload Execute Application Tags List No Tag Name Bit1 Bit3 c1 c2 Logic Trigger TagName Cojec TagName Cojec TagName Cojec T	_1 _2
Ac Nam Log Time	dd Updat e gic 1 ic Type er /Date Execute Lo Hour ur Minute 19 10 1	Timer Interval 10.00 Delete Timer Interval 10.00 Digic every Day of the we Sunday	ek	Program stat Loaded: 12, Cyclic Logic Yes Trigge	 Save Reload Execute Application Tags List No Tag Name Bit1 Bit3 C1 Bit3 C1 C2 C2 Logic Trigger TagName C Del Person_Counting Person_Reset PID1_Out1 PID1_Out1 PID1_Out1 PID1_SP PID1_STATUS PID2_Out1 PID3_DU2 PID3_DU2 PID4_DU2 PID4_DU2	_1 _2

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

"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.

Screen 1				
		Arduino		03/10/2022 17:42:40
Digital Inputs	Counters	Analog	lnputs	Analog Outputs
Presence Location #1	Reset	Communication	n counter proof	
Clear	9		10	
Presence Location #2 Person present	6	Thermistor 22.96 C	Temp LM35 23.44 C	
Hall magnetic sensor		73.33 F	74.19 F Reg6_ln [Volts]	Reg6_Out
Digital Outputs	SSB2 Fan	2.0	0.0 0.0 5.0 Reg7_In	0 0 255 Reg7_Out
	off	0.0	0	0
Remote CMD 0 CMD 1	CMD 2	0		

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".

	: List				Program edit	
No	Name	Type	Cyclic	Trigger M	if remote == 1	-
0	Logic1	Timer	Yes	Different	{	
1	Thermist	Timer	Yes	Different	Output2 = remote_cmd@0; Output4 = remote_cmd@1:	
2	Buzzer	Timer	Yes	Equal	Output5 = remote_cmd@2;	
3	Remote	Timer	No	Equal	if remote cmd2x=0 AND remote cmd2x=255	
5	PID	Timer	Yes	Equal	then Reg6_Out = remote_cmd2;	
					<pre>if remote_val2>=0 AND remote_val2<=255 then Reg7_Out = remote_val2; }</pre>	
•				F	4	+
A	dd Updat	Delete			Save Reload Execute Application Tags List	
Nam	e			Program statu	No Tag Name	
Ret	mote			Loaded: 6. P	repared: 6. Executed: 1 1 Bit1	Ĩ
1.00	ino ce			Loudedi off	2 Bit3	
					3 c1	۱ŀ
Log	ic Type	Timer Interval	[sec]	Cyclic Logic?	4 c2	1
Tim	er 🔻	5.00		No	▼ 5 Person_Counting_1	1
					6 Person_Counting_2	
					Logic Trigger TagName 7 Person_Reset	
	Data				8 PID1_CMD	
ime,	Date				9 PID1_Out1	
	Execute L	ogic every			Trigger Mode 10 PID 1_Out2	
	Hour	*			(when Tag Value) 11 PID1_SP	
	ir Minute	Day of the wee	ek		Equal I2 PID1_STATUS	
Hor	a minute	Day of the wee			13 PID2_CMD	
Hou		Sunday	×	Trigge	r Value Bit No 14 PID2_Out1	
Но ц	▼ 19 ▼					
Hou 00 Mor	▼ 19 ▼ 10 of the year	Day of the mor	nth	1	-1 -1 FID2_Out2	100

This program is executed 10 times per second.

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

7) Database setup

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

- Command
- Temperatures

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

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

After that we installed the MySQL driv ODBC access from our Windows com the remote MySQL database.

> Databases Queries Setup Queries List

> > No Database

MySQL3

Add

Trigger (when

Equal w

Status TagName

1 = connected; 3 = sql started; 5 = sql executed ok;

9 = sql done - no data or error; 16 = connection error;

Command Command TagName

-1 🔻

<< Del

<< Del

7

8

12

17

Person_Reset

PID1_CMD 9 PID1_Out1

10 PID1_Out2

13 PID2_CMD

14 PID2 Out1

15 PID2_Out2 16

PID2 SP

PID2_STATUS

11 PID1_SP PID1_STATUS

Trigger TagName

Trigger Value

1 = execute SOL;

Status

1

0

1

1

the	mysql> sl	now tables;	skup – I	FeSCAD	A 🕨 File	Edit Vie	
ase.	Tables	_in_storage	e				
	Alarms_ Alarms_ Commanc Tempera	_Active _History J atures					
b rage"	+ 4 rows in	n set (0.00	+ 0 sec)				
lage	mysql> de	escribe Cor	mmand;	pboard1	Clipbi		
	Field	∥Туре	Null	Key	Default	Extra	1
and 2	C1 C2 V1 V2	int(11) int(11) double double	YES YES YES YES	CADA2	NULL NULL NULL NULL		
updated	+ 4 rows in	n set (0.00	9 sec)	+	+	+	
	mysql> de	escribe Ter	mperatu	res;			
aluac	Field	Туре	Null	Key	Default	Extra	1
aiues	temp1 temp2	double double	YES		NULL NULL		
ers for	temp3 c1	double int(11)	YES		NULL		
outer to	CZ +	lnt(11) +	YES +	 +	NULL +	 +	+
OI Statem T	rigger Type	ODBC FI	leDSN name		Datab	ase access pa	ssword
PDATE Te T	imer only	MySQL3	3		123A	bCd#	
ELECT *F T	ïmer only	SQL Stat	tement				
		UPDATE temp1= temp2= temp3=	Temperatu #Temp#, #Adam_Reg #Light_sens	res set g4_Out#, sor#,			•
		Query Tr	igger Type		Query Int	erval [second:	s]
Update	Delete	Timer or	nly List	•	Database query	etatue:	
		No Tao Name			MySQL3 - Conn	ection OK	
	<< Del	1 Bit1					
Mode		2 Bit3		E			
ag Value)	-	3 c1					
th 👻	•	4 c2	har an Maria at			Destinatio	n Tags List
	Bit No	5 Person_C	ounting_1			No Tag N	lame
		- Croon C					

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.

14	Ρ	а	g	е
----	---	---	---	---

Add >>

Update >>

Delete

Save

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".

Querie	es List								
No	Database	SQL Statem	Trigger Type	-	ODBC FileDSN name		D <mark>at</mark> a	base ac	cess password
0	MySOL3	UPDATE Te	Timer only		MySQL3		123	AbCd#	
1	MySQL3	SELECT *F	Timer only		SQL Statement				
					SELECT * FROM Command;				
					Query Trigger Type		Query In	iterval [seconds]
•	m				Timer only	•	5	;	
	Add	Update	Delete	Appli	cation Tags List		Database quer	y status	8:
Trigge	er TagName			No	Tag Name	-	MySQL3 - Con	nection	OK - Reading (
			<< Del	1	Bit1				
	Tric	nger Mode		2	Bit3	Ξ			
	(wł	nen Tag Value)		3	c1				
	Eq	ual with	-	4	c2			De	stination Tags
				5	Person_Counting_1			No	Tag Name
Trigge	er Value		Bit No	6	Person_Counting_2				romata and
			-1 -	7	Person_Reset			2	remote_cmd
				8	PID1_CMD			2	remote val1
Comma	and Comma	nd TagName		9	PID1_Out1			4	remote val2
				10	PID1_Out2		Add >>	-	remote_vaiz
			< Del	11	PID1_SP				
1 = exe	ecute SQL;			12	PID1_STATUS		Update >>		
Status	Status	TagName		13	PID2_CMD				
1		-		14	PID2_Out1		Delete		
*				15					
L = con	nected; 3 =	sql started; 5 = sql	executed ok;	10	P102_5P		Save		

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 <u>https://www.noip.com</u>) 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
<pre>id="Panel"> <blockguote></blockguote></pre>
PANEL
<pre><button onclick="loadDoc('Command2.php', myFunction5)" type="button">READ </button></pre>
 <button onclick="sendDoc('Command3.php', myFunction6)" type="button">SEND </button> <label id="text1" readonly=""> </label>
<form ;"="" name="Commands" oninput="x_comm1.value = parseInt(comm1.value); y_comm2.value = parseInt(comm2.value); document.getElementById('text1').innerHTML = "></form>
Commands:
<input id="comm1" name="comm1" size="3" type="radio" value="0"/> Nothing
<input id="comm1" name="comm1" size="3" type="radio" value="1"/> Buzzer
<input id="comm1" name="comm1" size="3" type="radio" value="2"/> Light
<input id="comm1" name="comm1" size="3" type="radio" value="3"/> Light & Buzzer
<pre><input id="comm1" name="comm1" size="3" type="radio" value="6"/> Light & Fan</pre>
<pre><input id="comm1" name="comm1" size="3" type="radio" value="/"/> Light & Fan & Buzzer</pre>
<pre>0 <input id="comm2" max="255" min="0" name="comm2" type="range"/> 255 <output for="comm2" name="y_comm2"></output> <input id="x_comm1" name="x_comm1" type="hidden" value="0"/></pre>
 <button onclick="loadDoc('Values.txt', myFunction3)" type="button">Info </button> <pre>cpreadonly id="value"> </pre>
BACK
·······
<pre><script></script></pre>

```
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/>
br>Light sensor &nbsp = " + myVals.temp3 + " [\%]" +
"<br>br>Move counter#1 = " + myVals.c1 +
```

"
Move counter#2 = " + myVals.c2 ;
}

function myFunction5(xhttp) {
var myArduino = JSON.parse(xhttp.responseText);
document.forms["Commands"]["comm1"].value = myArduino.cmd1;
document.forms["Commands"]["x_comm1"].value = myArduino.cmd2;
document.forms["Commands"]["y_comm2"].value = myArduino.cmd2;
document.getElementById("text1").innerHTML = "";
}
function myFunction6(xhttp) {
document.getElementById("text1").innerHTML = xhttp.responseText;
}

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; $c^2 = -1;$ \$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->execute(); \$stmt->store_result(); \$stmt->bind_result(\$C1, \$C2, \$V1, \$V2); \$stmt->bind_result(\$C1, \$C2, \$V1, \$V2); \$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.



21 | Page