



Centro Universitário SENAI CIMATEC
Curso de Bacharelado em Engenharia Elétrica

Aplicação de comunicação MODBUS RTU/TCP com Arduino e software Elipse E3

**Cleber Couto Filho
Davi Costa**

Salvador-BA, 10 de agosto de 2018

Cleber Couto Filho

Davi Costa

Aplicação de comunicação MODBUS RTU/TCP com Arduino e software Elipse E3

Relatório apresentado como requisito parcial
para obtenção de aprovação na disciplina
Redes Industriais, no centro universitário
SENAI CIMATEC.

Docente: Alexandre Ribeiro

Coordenadora: Ana Beatriz Martins
Aguilar

Centro Universitário SENAI CIMATEC

Salvador-BA

10 de agosto de 2018

Lista de ilustrações

Figura 1 – Pacote da comunicação RTU	3
Figura 2 – Pacote da comunicação TCP	4
Figura 3 – Configuração do computador para comunicação TCP	13
Figura 4 – Etapa 1 da configuração RTU	14
Figura 5 – Etapa 2 da configuração RTU	14
Figura 6 – Etapa 3 da configuração RTU	15
Figura 7 – Etapa 1 da configuração TCP	15
Figura 8 – Etapa 2 da configuração TCP	16
Figura 9 – Etapa 3 da configuração TCP	16
Figura 10 – Driver RTU	17
Figura 11 – Driver TCP	17
Figura 12 – Tela de seleção do modo de comunicação	17
Figura 13 – Tela comunicação TCP	18
Figura 14 – Tela comunicação RTU	18

1 Introdução

A modernização dos processos faz com que seja necessário cada vez mais investimento em tecnologias de transmissão de informações e dados. Dispositivos como Controladores Lógicos Programáveis, placas microcontroladas como arduino e microcomputadores como o Raspberry são largamente aplicados e podem se utilizar das redes industriais para comunicação com dispositivos. O desenvolvimento das tecnologias de controle e comunicação possibilita para indústria a automatização cada vez mais efetivas de processos.

A atividade de desenvolver um supervisor que interligue o software Elipse E3 com o arduino consiste em uma aplicação prática de uma malha de comunicação, análogo ao que é encontrado normalmente na indústria. O arduino é um dispositivo que está se popularizando cada vez mais, não sendo a melhor solução em quesitos de preço e velocidade, mas se mostra prático e didático, assim conseguindo ilustrar os conhecimentos discutidos no decorrer da disciplina.

1.1 Proposta

Desenvolver aplicação no software Elipse E3, que se comunica com a placa microcontrolada Arduino, utilizando os protocolos de comunicação MODBUS RTU e MODBUS TCP.

1.1.1 Requisitos do Projeto

- Apresentar uma interface gráfica no Software Elipse E3 para comunicação com o usuário e display de informações
- Receber um valor de um número decimal, converter para binário e mostrar no sistema de LEDS e na interface;
- Apresentar status de botão conectado ao Arduino;
- Fazer leitura do valor de um potenciômetro, adicionar um *offset* fornecido pelo usuário e apresentar na interface;

1.2 Fundamentação teórica

Para realização do trabalho foram necessários os seguintes conhecimentos relativos à redes e comunicações.

1.2.1 Comunicação Serial

Comunicação serial é a nomenclatura atribuída ao processo da troca de dados de forma sequencial por meio de um canal ou barramento específico para comunicação. Tais dados são bytes de informação, transmitidos bit a bit, por meio de uma porta serial.

Os protocolos de comunicação serial representam o modo que a mensagem é transmitida, definindo a forma como os bytes serão ordenados de modo a garantir que a mensagem seja transmitida. O padrão de comunicação diz respeito à estrutura física da comunicação, fazendo referência aos padrões elétricos envolvidos e quantidade de vias utilizadas.

1.2.2 Protocolo MODBUS

O protocolo MODBUS O protocolo Modbus é uma estrutura de mensagem aberta desenvolvida pela Modicon na década de 70. De acordo com a MODBUS Organization o protocolo MODBUS É: “É um protocolo de camada de aplicação posicionado no nível 7 do modelo OSI, que fornece comunicação cliente / servidor entre dispositivos conectados em diferentes tipos de barramentos ou redes”.

Consiste em um protocolo de solicitação/resposta, fornecendo serviços especificados por códigos de função. Os códigos de função de MODBUS são elementos das PDUs (Protocol Data Unit ou unidade de dados de protocolo) de solicitação / resposta de MODBUS. É largamente utilizado em automação industrial, sendo transmitido por protocolos físicos como o RS232/RS485 e Ethernet TCP/IP.

1.2.2.1 Modo de transmissão RTU

RTU é a sigla para Remote Terminal Unit, nesse modo de transmissão cada mensagem de dados de 8 bits contém dois caracteres hexadecimais de 4 bits, isso faz com que o processamento de dados seja mais rápido se comparado com a transmissão por 2 caracteres Ascii de 4 bits, já que a densidade de caracteres é maior. Apresenta 8 bits disponíveis para o endereço, 8 para função, 0 a 252 bits de data e 16 bits para o CRC Check, como mostrado na figura 1 a seguir:

Address	Function	Data	CRC Check
8 bits	8 bits	N x 8 bits	16 bits

Figura 1 – Pacote da comunicação RTU

1.2.2.2 Modo de transmissão TCP

O modo de transmissão TCP é uma aplicação do protocolo MODBUS baseado em TCP/IP, consequentemente utilizando a conexão ethernet. Para a pilha de comunicação, nesse protocolo é adicionado ao quadro um cabeçalho MBAP (MODBUS Application Protocol), deitando o modelo de mensagem como a figura 2:

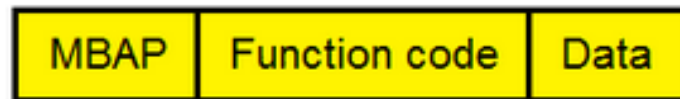


Figura 2 – Pacote da comunicação TCP

Esse novo cabeçalho tem 7 bytes de tamanho sendo composto por os seguintes elementos:

- Transaction identifier: usado para identificação da resposta para a transação (2 bytes);
- Protocol identifier: 0 (zero) indica Modbus (2 bytes);
- Length: contagem de todos os próximos bytes (2 bytes);
- Unit identifier: utilizado para identificar o escravo remoto em uma rede Modbus RTU (1 byte).

O Modbus TCP não utiliza um byte de checagem ao final da mensagem, pois o próprio frame ethernet já apresenta uma checagem do tipo CRC-32. Na comunicação o cliente Modbus TCP inicia a conexão com o servidor para enviar as requisições, onde a porta padrão para a conexão com os servidores é a TCP 502.

2 Metodologia e Resultados

2.1 Desenvolvimento do código Arduino RTU

O programa realiza a conversão de valores decimais em binários, lê o status de um botão e o valor de um potenciômetro somado com um setpoint.

Para aplicação do modo de transmissão MODBUS RTU no Arduino, foi utilizada a biblioteca "SimpleModbusSlave.h", onde o arduino foi configurado como escravo, sendo assim, o computador o mestre.

A função `modbus_configure` é responsável pela configuração dos parâmetros da comunicação, foram escolhidos os valores de baud-rate sendo iguais a 9600, o formato de dados como `SERIAL_N2`, que equivale ao tamanho da palavra de dados, e o endereço do arduino como sendo 1.

O Arduino se comunica com o usuário por meio da interface gráfica, dessa forma as variáveis de entrada do programa são o valor para ser convertido em binário, um *setpoint* para ser adicionado ao valor do potenciômetro lido e o sinal de botão pressionado, proveniente de um botão físico.

O programa possui 3 saídas: o valor do número decimal convertido para binário e mostrado num sistema físico de 3 leds; o mesmo valor já convertido para binário mostrado também na interface gráfica e o valor lido pelo potenciômetro mais o *setpoint* informado. O programa está mostrado à seguir:

```
1 #include <SimpleModbusSlave.h>
2
3 int LED1 = 8; //Definicao das portas dos LEDs
4 int LED2 = 6;
5 int LED3 = 4;
6 int botao = 2;
7
8
9
10 enum //Enumera as variaveis que serao utilizadas para
      comunicacao com o Elipse E3
11 {
12 BINARIO, // recebe o valor em decimal para ser convertido em
      binario
13 LED_1,
```

```
14 LED_2,
15 LED_3,
16 POTENCIOMETRO, //Recebe o valor do sensor LDR; primeiro
    componente que o E3 ira "\"reconhecer\""(N4 = 01)
17 SETPOINT, //Recebe o valor decimal; (N4 = 02)
18 SETPOT,
19 BOTAO,
20 HOLDING_REGS_SIZE //Identifica a quantidade de holdingRegs que
    estao sendo utilizados no programa.
21 };
22
23 unsigned int holdingRegs[HOLDING_REGS_SIZE]; //Variavel criada
    para manipulacao dos registradores que foram criados.
24
25 int num = 0;
26 int setpoint;
27
28 void setup() {
29
30 modbus_configure(&Serial, 9600, SERIAL_8N2, 1, 2,
    HOLDING_REGS_SIZE, holdingRegs); //Determina os parametros
    necessarios para estabelecer a conexao via comunicacao
    serial utilizando MODBUS.
31 //9600 = velocidade da transmissao dos dados; SERIAL_8N2 =
    formato do pacote utilizado no MODBUS; 1 = identificacao do
    escravo.
32
33 modbus_update_comms(9600, SERIAL_8N2, 1); //Funcao tambem
    responsavel pela comunicacao via MODBUS;
34
35 pinMode (LED1, OUTPUT); //Modos de operacao dos LEDs (saida)
36 pinMode (LED2, OUTPUT);
37 pinMode (LED3, OUTPUT);
38 pinMode (botao, INPUT);
39 }
40
41
42 void loop() {
43
```



```
44 modbus_update(); // Funcao utilizada para a atualizacao dos
    valores dos registradores declarados (V_LDR, V_LED...)
45
46 holdingRegs[POTENCIOMETRO] = analogRead(A0); //Le-se a
    informacao presente na porta analogica A0 (Sensor LDR)
47 setpoint = holdingRegs[SETPOINT];
48 holdingRegs[SETPOT] = setpoint + analogRead(A0);
49 holdingRegs[BOTAO] = digitalRead(botao);
50
51 switch(holdingRegs[BINARIO]) {
52 case 0:
53     digitalWrite(LED1, LOW);
54     digitalWrite(LED2, LOW);
55     digitalWrite(LED3, LOW);
56     holdingRegs[LED_1] = 0;
57     holdingRegs[LED_2] = 0;
58     holdingRegs[LED_3] = 0;
59     break;
60 case 1:
61     digitalWrite(LED1, HIGH);
62     digitalWrite(LED2, LOW);
63     digitalWrite(LED3, LOW);
64     holdingRegs[LED_1] = 1;
65     holdingRegs[LED_2] = 0;
66     holdingRegs[LED_3] = 0;
67     break;
68 case 2:
69     digitalWrite(LED1, LOW);
70     digitalWrite(LED2, HIGH);
71     digitalWrite(LED3, LOW);
72     holdingRegs[LED_1] = 0;
73     holdingRegs[LED_2] = 1;
74     holdingRegs[LED_3] = 0;
75     break;
76 case 3:
77     digitalWrite(LED1, HIGH);
78     digitalWrite(LED2, HIGH);
79     digitalWrite(LED3, LOW);
80     holdingRegs[LED_1] = 1;
```

```
81 holdingRegs[LED_2] = 1;
82 holdingRegs[LED_3] = 0;
83 break;
84 case 4:
85 digitalWrite(LED1, LOW);
86 digitalWrite(LED2, LOW);
87 digitalWrite(LED3, HIGH);
88 holdingRegs[LED_1] = 0;
89 holdingRegs[LED_2] = 0;
90 holdingRegs[LED_3] = 1;
91 break;
92 case 5:
93 digitalWrite(LED1, HIGH);
94 digitalWrite(LED2, LOW);
95 digitalWrite(LED3, HIGH);
96 holdingRegs[LED_1] = 1;
97 holdingRegs[LED_2] = 0;
98 holdingRegs[LED_3] = 1;
99 break;
100 case 6:
101 digitalWrite(LED1, LOW);
102 digitalWrite(LED2, HIGH);
103 digitalWrite(LED3, HIGH);
104 holdingRegs[LED_1] = 0;
105 holdingRegs[LED_2] = 1;
106 holdingRegs[LED_3] = 1;
107 break;
108 case 7:
109 digitalWrite(LED1, HIGH);
110 digitalWrite(LED2, HIGH);
111 digitalWrite(LED3, HIGH);
112 holdingRegs[LED_1] = 1;
113 holdingRegs[LED_2] = 1;
114 holdingRegs[LED_3] = 1;
115 break;
116 }
117 }
```

2.2 Desenvolvimento do código Arduino TCP

Assim como o código anterior, este programa realiza a conversão de valores decimais em binários, lê o status de um botão e o valor de um potenciômetro somado com um *setpoint*, a comunicação entre o Arduino e Eclipse é feita por meio do protocolo TCP. Para aplicação do modo de transmissão MODBUS TCP no arduino, foram utilizadas as bibliotecas "SPI.h", "Ethernet.h" e "Mudbus.h".

A biblioteca "SPI.h" é responsável pela comunicação serial entre o *shield* Ethernet e Arduino, a biblioteca "Ethernet.h" realiza a configuração dos parâmetros de ethernet e pela conectividade a rede e a biblioteca "Mudbus.h" configura protocolo modbus da comunicação. Para a comunicação o arduino foi configurado como servidor, recebendo as requisições do computador, que funciona como cliente.

A função `Ethernet.begin` é responsável pela configuração dos parâmetros a comunicação, foram escolhidos os valores de IP sendo igual 192.168.1.1, o gateway igual a 192.168.1.1 e endereço de subrede de 255.255.255.0.

O programa está ilustrado a seguir, apresentando as mesmas entradas e saídas do anterior.

```
1 #include <SPI.h>
2 #include <Ethernet.h>
3 #include "\"Mudbus.h\"
4
5 Mudbus Mb;
6 //Function codes 1(read coils), 3(read registers), 5(write coil
   ), 6(write register)
7 //signed int Mb.R[0 to 125] and bool Mb.C[0 to 128] MB_N_R
   MB_N_C
8 //Port 502 (defined in Mudbus.h) MB_PORT
9
10 int botao = 8;
11 int LED1 = 7; //Definicao das portas dos LEDs
12 int LED2 = 6;
13 int LED3 = 5;
14 int setpoint;
15
16
17 void setup() {
18 uint8_t mac[]      = { 0x90, 0xA2, 0xDA, 0x00, 0x51, 0x06 };
19 uint8_t ip[]       = { 192, 168, 1, 10 };
```

```
20 uint8_t gateway[] = { 192, 168, 1, 1 };
21 uint8_t subnet[]  = { 255, 255, 255, 0 };
22 Ethernet.begin(mac, ip, gateway, subnet);
23 //With the last update of Industrial Shields boards it\'s not
    necessary to use function pinMode()
24
25
26 pinMode (LED1, OUTPUT); //Modos de operacao dos LEDs (saida)
27 pinMode (LED2, OUTPUT);
28 pinMode (LED3, OUTPUT);
29 pinMode (botao, INPUT);
30 }
31
32 void loop() {
33 Mb.Run(); //Update the values of Mb.R and Mb.C every loop cycle
34
35 switch(Mb.R[0]) {
36 case 0:
37     digitalWrite(LED1, LOW);
38     digitalWrite(LED2, LOW);
39     digitalWrite(LED3, LOW);
40     Mb.R[1] = 0;
41     Mb.R[2] = 0;
42     Mb.R[3] = 0;
43     break;
44 case 1:
45     digitalWrite(LED1, HIGH);
46     digitalWrite(LED2, LOW);
47     digitalWrite(LED3, LOW);
48     Mb.R[1] = 1;
49     Mb.R[2] = 0;
50     Mb.R[3] = 0;
51     break;
52 case 2:
53     digitalWrite(LED1, LOW);
54     digitalWrite(LED2, HIGH);
55     digitalWrite(LED3, LOW);
56     Mb.R[1] = 0;
57     Mb.R[2] = 1;
```

```
58 Mb.R[3] = 0;
59 break;
60 case 3:
61 digitalWrite(LED1, HIGH);
62 digitalWrite(LED2, HIGH);
63 digitalWrite(LED3, LOW);
64 Mb.R[1] = 1;
65 Mb.R[2] = 1;
66 Mb.R[3] = 0;
67 break;
68 case 4:
69 digitalWrite(LED1, LOW);
70 digitalWrite(LED2, LOW);
71 digitalWrite(LED3, HIGH);
72 Mb.R[1] = 0;
73 Mb.R[2] = 0;
74 Mb.R[3] = 1;
75 break;
76 case 5:
77 digitalWrite(LED1, HIGH);
78 digitalWrite(LED2, LOW);
79 digitalWrite(LED3, HIGH);
80 Mb.R[1] = 1;
81 Mb.R[2] = 0;
82 Mb.R[3] = 1;
83 break;
84 case 6:
85 digitalWrite(LED1, LOW);
86 digitalWrite(LED2, HIGH);
87 digitalWrite(LED3, HIGH);
88 Mb.R[1] = 0;
89 Mb.R[2] = 1;
90 Mb.R[3] = 1;
91 break;
92 case 7:
93 digitalWrite(LED1, HIGH);
94 digitalWrite(LED2, HIGH);
95 digitalWrite(LED3, HIGH);
96 Mb.R[1] = 1;
```

```
97 Mb.R[2] = 1;
98 Mb.R[3] = 1;
99 break;
100 }
101
102
103 Mb.R[4] = analogRead(A0);
104 setpoint = Mb.R[5];
105
106 Mb.R[6] = Mb.R[4] + setpoint;
107
108 // if( digitalRead(botao)) {
109 //
110 //     Mb.R[7] = 1;
111 // }
112 // else{
113 //     Mb.R[7] = 0;
114 // }
115 Mb.R[7] = digitalRead(botao);
116 }
```

2.3 Configuração do computador

Para comunicar o computador e o Arduino via Ethernet é preciso criar uma rede local entre eles, como mostrado anteriormente, o Arduino foi configurado para um IP de 192.168.1.10 e sub-máscara de 255.255.255.00, o rede do computador foi configurado com o IP 192.168.1.23 com máscara 255.255.255.0 o mesmo estara na mesma rede com o Arduino e assim podendo trocar informações.

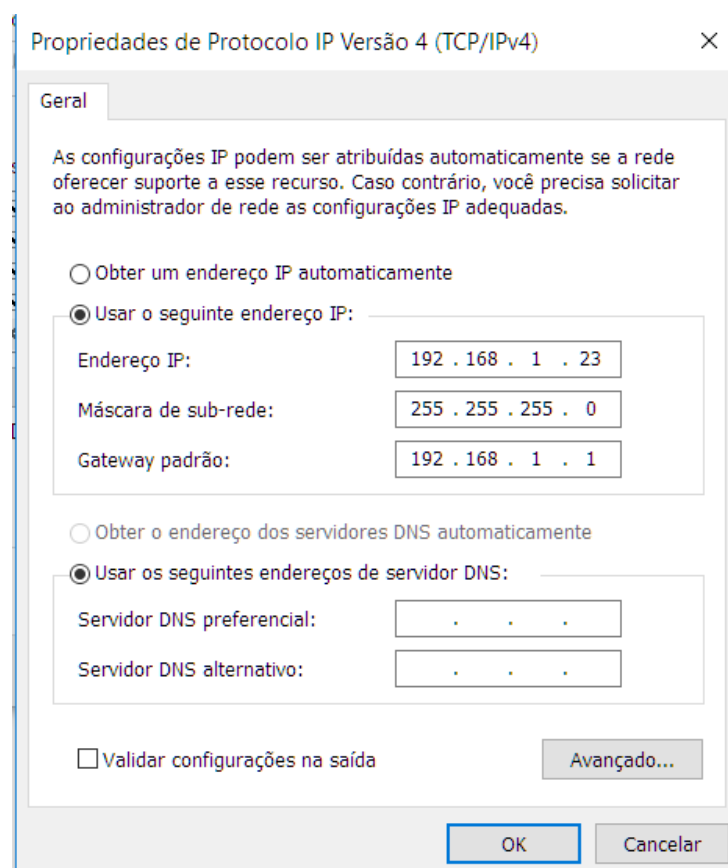


Figura 3 – Configuração do computador para comunicação TCP

2.4 Comunicação Elipse/Arduino utilizando o modo RTU

Para realizar a comunicação entre o elipse e o arduino foi necessário o uso de um *Driver* para as duas aplicações, no modo RTU e TCP. Ao abrir a janela de configuração do Driver RTU, foram adicionados duas funções contendo como mostra na figura 4, foi selecionado também que o Modbus mode seria RTU. Na aba *setup*, ilustrado na figura 5, foi selecionado a camada física “Serial” e o restante Default. As etapa da configuração estão mostradas a seguir:

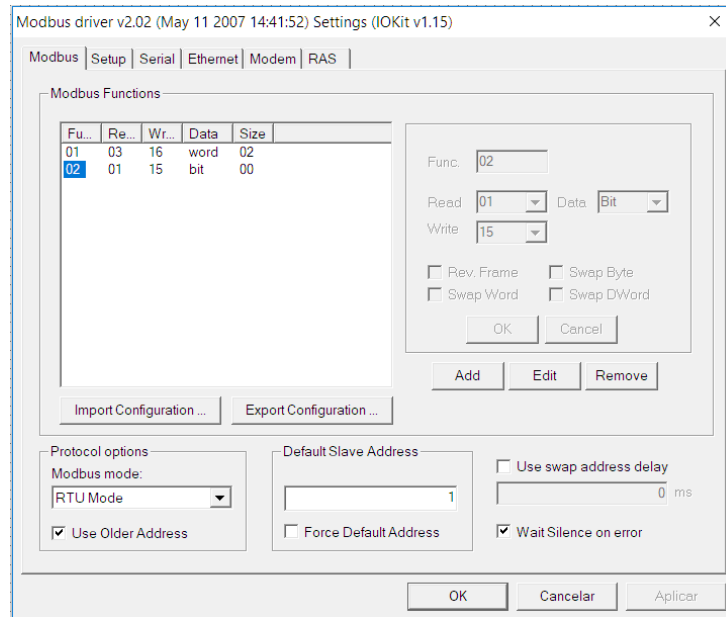


Figura 4 – Etapa 1 da configuração RTU

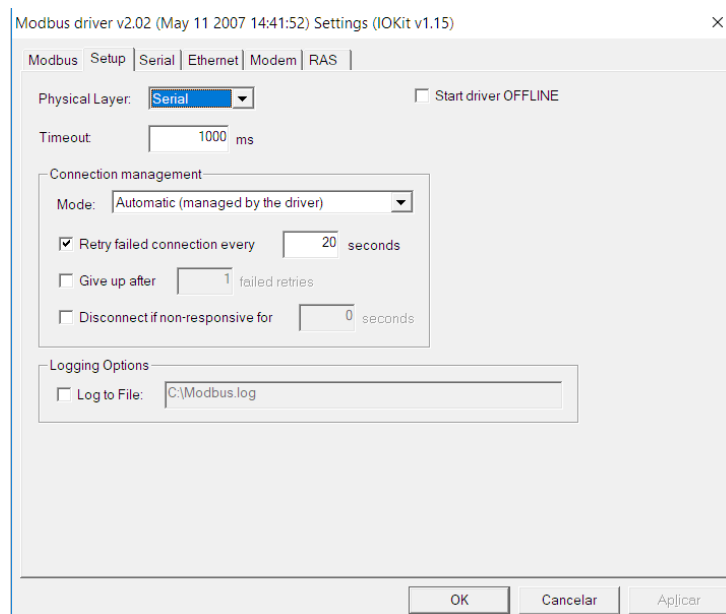


Figura 5 – Etapa 2 da configuração RTU

Após selecionar a camada física Serial, a aba serial pode ser configurada. Nela é configurado a *com port* referente a com na qual o Arduino esta conectado, juntamente com a configuração de *baud-rate* para 9600, data bits para 8, bit paridade “nenhum e o *stop* bits para 1, como foi configurado no código Arduino.

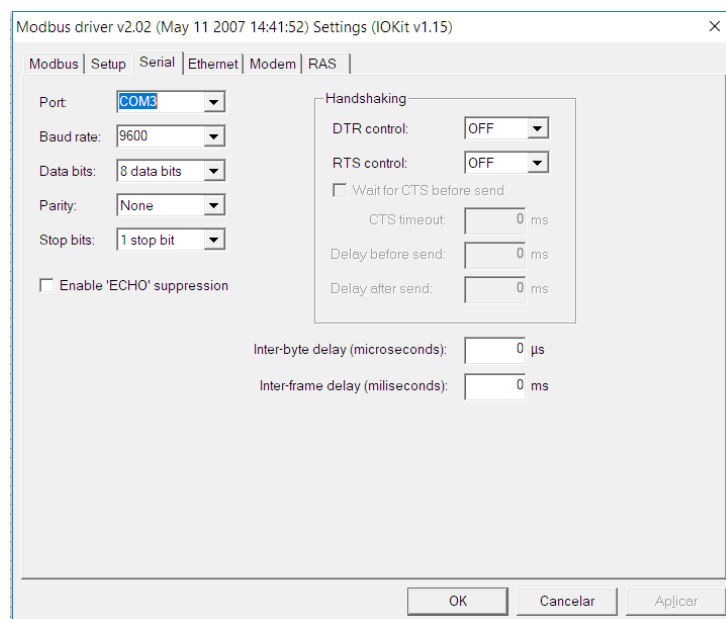


Figura 6 – Etapa 3 da configuração RTU

2.5 Comunicação Eclipse/Arduino utilizando o modo TCP

O driver TCP foi configurado de forma parecida ao modo RTU, na aba Modbus única diferença é o modo modbus para "modbus TCP", como mostra a figura 7:

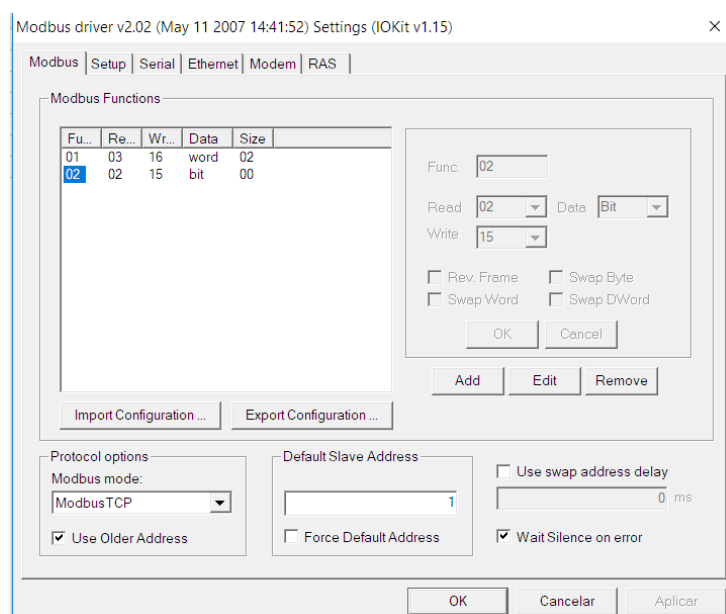


Figura 7 – Etapa 1 da configuração TCP

Na aba "setup", a camada fisica foi selecionada "ethernet", como mostra a figura 8:

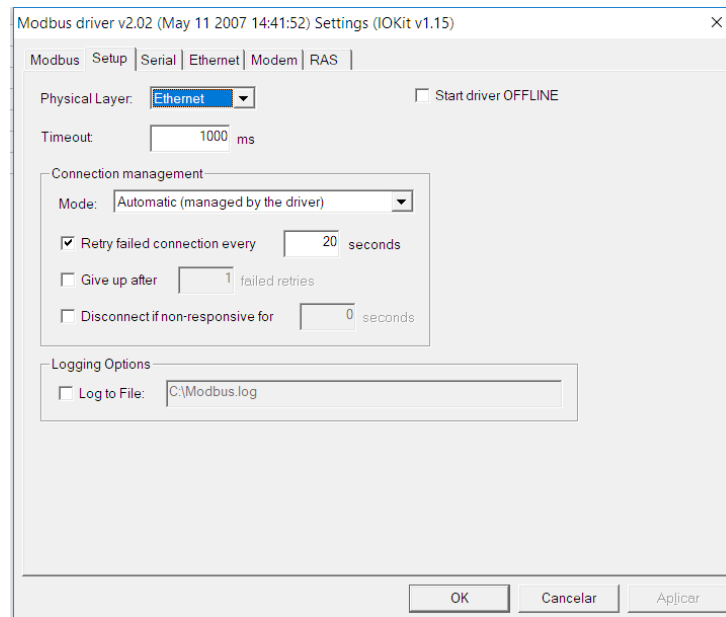


Figura 8 – Etapa 2 da configuração TCP

Na aba “ethernet”, no campo “transport” foi escolhido TCP/IP, nos campo IP e *port* foram colocados os mesmos valores configurados no arduino.

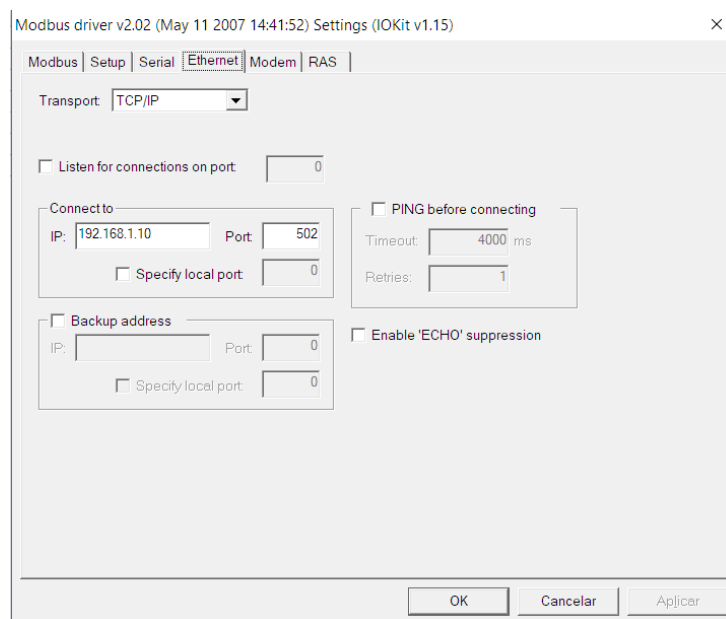


Figura 9 – Etapa 3 da configuração TCP

2.6 Configuração das *tags* no Elipse

Para cada driver, a configuração das tags de comunicação se diferenciou pelo os IDs de cada variável que é representado nas figuras à seguir como a coluna *P4*. A coluna *P1* representa a configuração do escravo que participará da comunicação, onde na comunicação com o arduino, se adotou o valor de $P1 = 1$. *P2* mostra a função utilizada na

configuração do driver, $P3$ indica a utilização de memória estendida. Foi utilizada a função 1 de HoldingRegs, assim $P2 = 1$ e não se utilizará a memória estendida, assim $P3 = 0$.

Nome	Disp...	Item	P1/N1/B1	P2/N2/B2	P3/N3/B3	P4/N4/B4	Ta...	Varredura	Leitur...	Escrit...	Escala?	Mín. ...	Máx. ...	UE	Mín. ...	Máx. ...
DriverRTU			0	0	0	0										
• binario			1	1	0	1		10	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	0	1000		0	1000
• led1			1	1	0	2		10	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	0	1000		0	1000
• led2			1	1	0	3		10	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	0	1000		0	1000
• led3			1	1	0	4		10	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	0	1000		0	1000
• potenciome			1	1	0	5		500	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	0	1000		0	1000
• setpoint			1	1	0	6		10	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	0	1000		0	1000
• pot+setp			1	1	0	7		500	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	0	1000		0	1000
• botao			1	1	0	8		10	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	0	1000		0	1000

Figura 10 – Driver RTU

Nome	Disp...	Item	P1/N1/B1	P2/N2/B2	P3/N3/B3	P4/N4/...	Ta...	Varredura	Leitura?	Escrita?	Escala?	Mín. ...	Máx. ...	UE	Mín. ...	Máx. ...
DriverTCP			0	0	0	0										
• binario			1	1	0	1		10	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	0	1000		0	1000
• led1			1	1	0	2		10	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	0	1000		0	1000
• led2			1	1	0	3		10	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	0	1000		0	1000
• led3			1	1	0	4		10	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	0	1000		0	1000
• potenciome			1	1	0	5		500	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	0	1000		0	1000
• setpoint			1	1	0	6		10	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	0	1000		0	1000
• pot+setp			1	1	0	7		500	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	0	1000		0	1000
• botao			1	1	0	8		10	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	0	1000		0	1000

Figura 11 – Driver TCP

2.7 Interface Gráfica

A interface gráfica recebe o número para ser convertido em binário e o *offset* que será adicionado ao valor do potenciômetro. Cada campo foi relacionado com as *tags* de comunicação do driver mostradas anteriormente, buscou-se desenvolver uma interface simples, os *screenshots* estão mostrados à seguir:

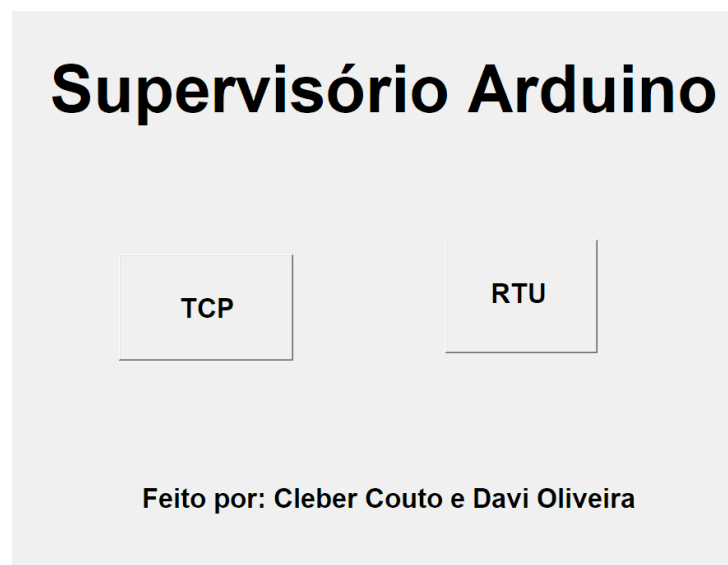


Figura 12 – Tela de seleção do modo de comunicação

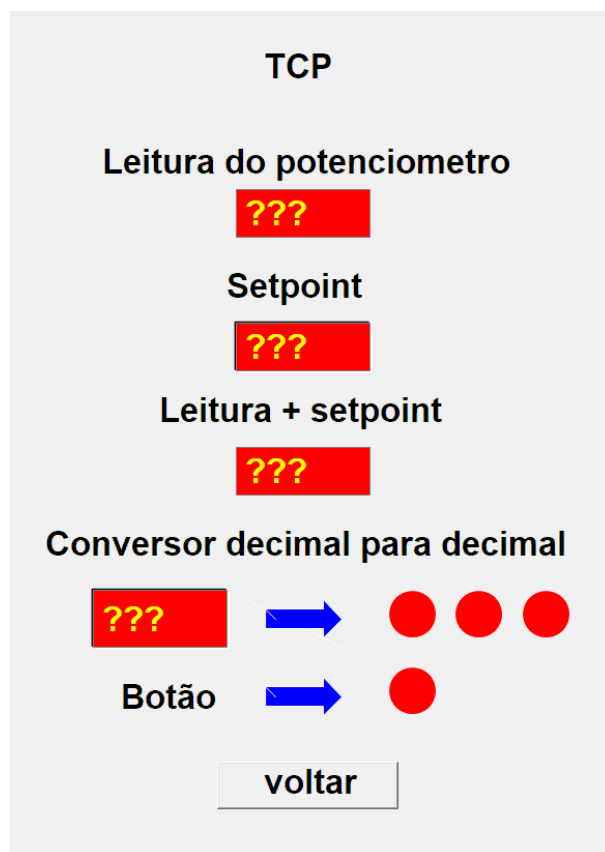


Figura 13 – Tela comunicação TCP

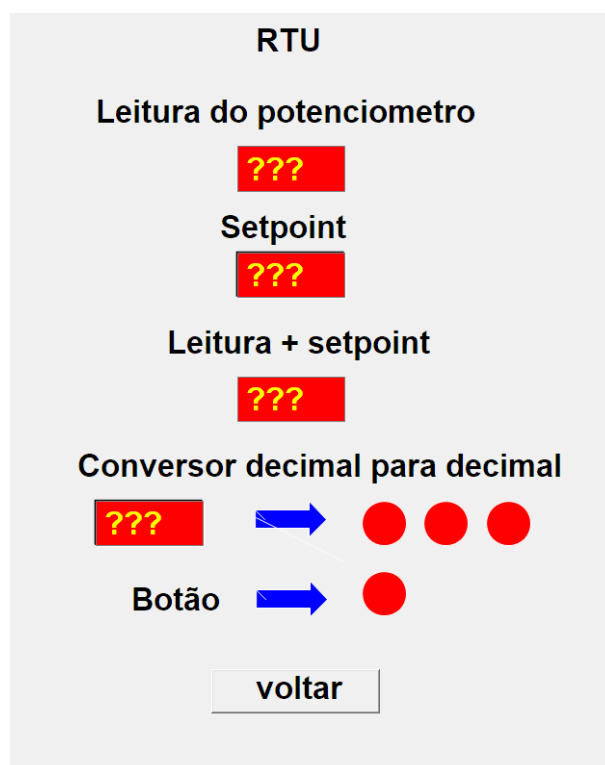


Figura 14 – Tela comunicação RTU

Referências

- [1] MODBUS, I. D. A. Modbus application protocol specification v1. 1a. North Grafton, Massachusetts (www.modbus.org/specs.php), 2004.
- [2] Protocolo MODBUS - Disponível em: <https://www.embarcados.com.br/protocolo-modbus/>
- [3] Modbus RTU Unplugged - Disponível em: <https://www.rtaautomation.com/technologies/modbus-rtu/>