UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Uso do software HOME I/O e CONNECT I/O para controle de residência simulada usando Modbus e como controlador um Arduino

RELATÓRIO DA DISCIPLINA INTEGRAÇÃO DE REDES INDUSTRIAIS
Prof. Frederico Schaf

Daniel Schreiner
Eduardo Tier
João Renato Amado
Nícolas Eugênio Lima Basquera

Santa Maria, RS, Brasil

2017

SUMÁRIO

CAPÍTULO 1 INTRODUÇÃO

Os sistemas de controle e automação industriais englobam uma grande variedade de paradigmas, comunicação e tecnologias de computação. Diversos fornecedores tentam garantir a interoperabilidade de seus componentes de forma conjunta entre eles e o nível de complexidade tende a aumentar para especificação e projeto.

O Modbus é um protocolo aberto muito utilizado em redes industriais. O Modbus TCP mestre no arduíno, desenvolvido neste trabalho, pela pesquisa que o grupo fez, ainda não é muito difundido na rede e possui pouco material e trabalhos implementados. Frente a essa dificuldade, o seguinte trabalho propõe a escrita de funções que possam ser usadas no Modbus TCP Mestre (cliente), para conectar um arduíno com o software HOME I/O através do CONNECT I/O, que funcionada como escravo (servidor), para simular o controle de variáveis presentes na casa.

CAPÍTULO 2 PROTOCOLO MODBUS

O protocolo Modbus é uma estrutura de mensagem aberta desenvolvida pela Modicon na década de 70, utilizada para comunicação entre dispositivos mestre-escravo / cliente-servidor. A Modicon foi posteriormente adquirida pela Schneider e os direitos sobre o protocolo foram liberados pela Organização Modbus. Muitos equipamentos industriais utilizam o Modbus como protocolo de comunicação, e graças às suas características, este protocolo também tem sido utilizado em uma vasta gama de aplicações

Neste trabalho, será utilizada a variação **Modbus TCP/IP**. Os dados são encapsulados em quadros ethernet, pacotes IP e segmentos TCP. Utiliza a porta 502 da pilha TCP/IP em velocidade compatível com o TCP. Dentro do Modbus, existem tipos de variáveis: “Discrete Inputs”, “Coils”, “Input Registers” e “Holding Registers”.

Os códigos de função (requisição de serviços) mais comuns são:

- 01 – Read coil status – leitura de múltiplas coils
- 02 – Read input status – Leitura de múltiplos discrete inputs
- 03 – Read holding registers – leitura de múltiplos holding registers.
- 04 – Read input register – leitura de múltiplas input registers
- 05 – Write coil – escrita de uma única coil
- 06 – Write single register – escrita em um único holding register

2.1 Estrutura do protocolo

Esta seção irá descrever a forma geral do encapsulamento da requisição Modbus quando a variação Modbus TCP é utilizada. Todas as requisições o Modbus TCP são feitas através da porta 502. A requisição e a resposta possuem 6 bytes pré-definidos como segue:

- **Byte 0**: Identificador da transação: Normalmente recebe **0**
- **Byte 1**: Identificador da transação: Normalmente recebe **0**
- **Byte 2**: Identificador do protocolo = **0**
- **Byte 3**: Identificador do protocolo = **0**
- **Byte 4**: Comprimento do quadro (endereço alto) = **0** (já que as mensagens são menores que 256)
- **Byte 5**: Comprimento do quadro (endereço alto) = **número de bytes que seguem**

Os bytes seguintes são variáveis e podem ser resumidos a:

- **Byte 6**: ID do escravo
- **Byte 7**: Código de função do Modbus
- **Byte 8**: Dados conforme necessário

2.1.1 Read Input Registers

Para que seja possível ler os input registers, os bytes 0 até 5 são iguais conforme já descritos. Os bytes seguintes seguem a seguinte estrutura descrita a seguir para a requisição.

- Byte 6: Endereço do escravo
- Byte 7: FC = **0x04**
- Byte 8: Endereço Alto da primeira input register a ser lida
- Byte 9: Endereço Baixo da primeira input register a ser lida
- Byte 10: 0x00 (bytes nível alto da quantidade de holding regs. a serem lidos)
- Byte 11: 0x01 (bytes nível baixo da quantidade de holding regs. a serem lidos)

É importante notar que apenas **1** holding register será lido por vez, mesmo o protocolo permitindo uma leitura múltipla. A resposta terá o seguinte formato:

- Byte 6: ID do escravo
- Byte 7: FC = **0x04**
- Byte 8: Número de input registers lidos
- Byte 9 em diante: Valores dos input registers

2.1.2 Read Input Discretes

Para que seja possível ler as input discretes, os bytes de 0 a 5 já descritos continuam os mesmos. Os bytes seguintes tem a seguinte estrutura:

- Byte 6: ID do escravo
- Byte 7: FC = **0x02**
- Byte 8: Endereço Alto da primeira input discrete a ser lida
- Byte 9: Endereço Baixo da primeira input discrete a ser lida
- Byte 10: 0x00 (bytes nível alto da quantidade de input discretes a serem lidas)
- Byte 11: 0x01 (bytes nível baixo da quantidade de input discretes a serem lidas)

A resposta tem o seguinte formato:

- Byte 6: ID do escravo
- Byte 7: FC = **0x02**
- Byte 8: Contador dos bytes seguintes
- Byte 9: Valor do bit lido do primeiro endereço solicitado

Novamente, neste caso somente 1 input discrete será lida por vez, então o byte 9 já irá conter o valor da variável solicitada.

2.1.3 Write coil

Na escrita das coils, os bytes de 0 até 5 continuam os mesmos como já definidos. Os seguintes bytes apresentam a seguinte estrutura:

- Byte 6: ID do escravo
- Byte 7: FC = **0x05**
- Byte 8: Endereço Alto da primeira coil
- Byte 9: Endereço Baixo da primeira coil
- Byte 10: =FF (para coil = 1), =00 (para coil = 0)
- Byte 11: =00

Resposta:

- Byte 6: ID do escravo
- Byte 7: FC = **0x05**
- Byte 8: =FF (para coil = 1), =00 (para coil = 0)
- Byte 11: =00

2.1.4 Write single holding register

Na escrita do holding register, os bytes de 0 até 5 continuam os mesmos como já definidos. Os seguintes bytes apresentam a seguinte estrutura:

- Byte 6: ID do escravo
- Byte 7: FC = **0x06**
- Byte 8: Endereço Alto do holding register
- Byte 9: Endereço Baixo do holding register

- Byte 10: Endereço alto do valor desejado
- Byte 11: Endereço baixo do valor desejado

Resposta:

- Byte 6: ID do escravo
- Byte 7: FC = **0x06**
- Byte 8: Endereço Alto do holding register
- Byte 9: Endereço Baixo do holding register
- Byte 10: Endereço alto do valor determinado
- Byte 11: Endereço baixo do valor determinado

CAPÍTULO 3 SHIELD ETHERNET

Um shield ethernet permite conectar uma placa Arduíno a uma rede ethernet de forma que suas funções possam ser controladas pela internet. Este Ethernet Shield baseia-se no chip WIZnet ethernet W5100 que fornece acesso à rede (IP) nos protocolos TCP ou UDP e é facilmente utilizado usando as bibliotecas Ethernet Library e SD Library.

Ele é compatível tanto com o Arduíno Uno e Mega e possui um slot para cartão micro-SD que pode ser usado para armazenar arquivos que vão servir na rede.

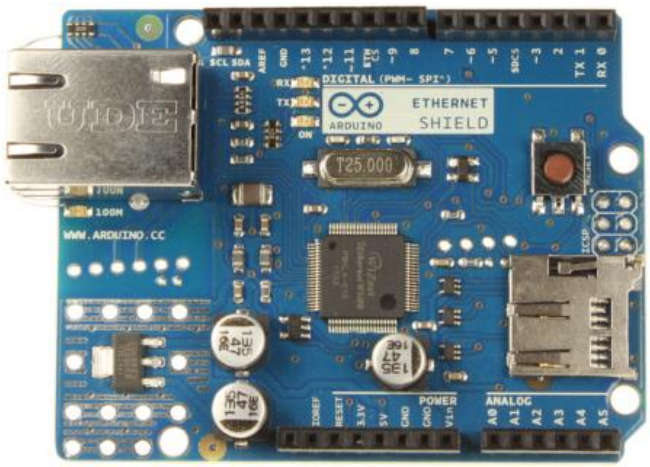

Figura 1 - Shield Ethernet

CAPÍTULO 4 HOME I/O E CONNECT I/O

O HOME I/O é software de simulação interativa de uma casa inteligente. Com este software, conforme o fabricante, os estudantes podem aprender sobre automação, comportamentos térmicos, consumo de energia, etc. O objetivo principal do software é introduzir conceitos de automação usando uma interativa casa inteligente.

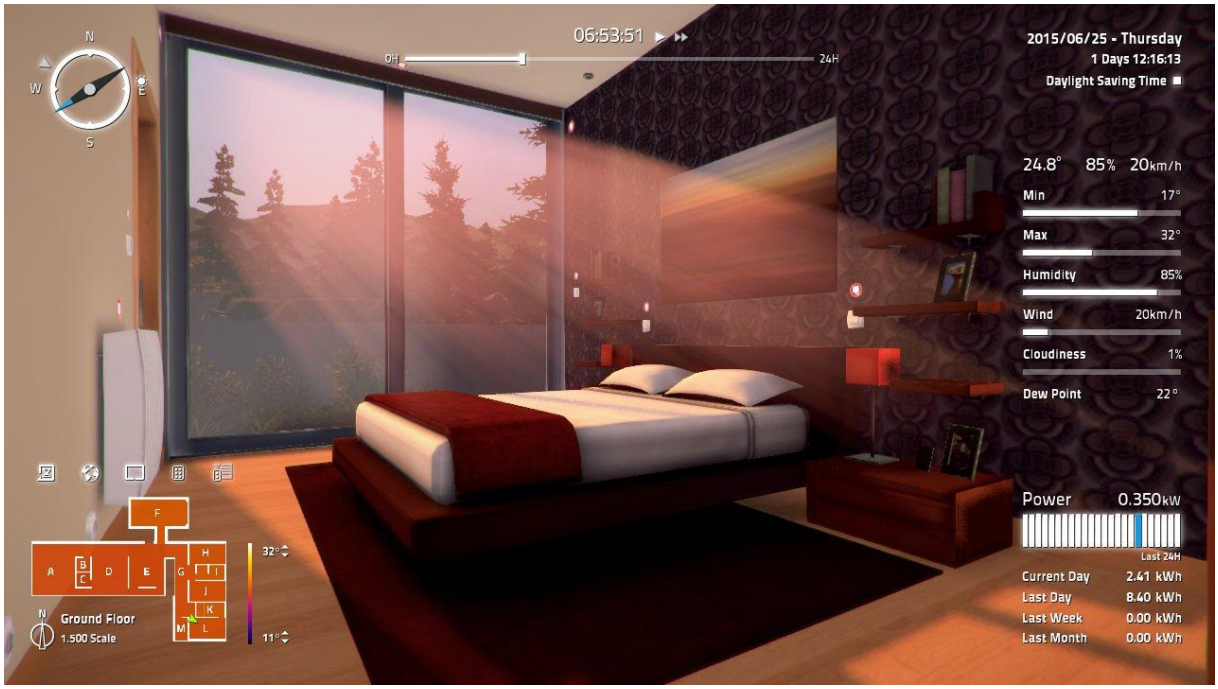

Figura 2 - Apresentação HOME I/O

Como pode ser visto no canto inferior esquerdo da figura anterior, a casa é dividida em salas com letras correspondentes a cada cômodo. O CONNECT I/O é uma ferramenta presente no HOME I/O que implementa determinadas funcionalidades desenhando diagramas e conectando nodos de forma interativa. Através do CONNECT I/O, será possível criar um servidor Modbus TCP que funcionará como escravo para o sistema proposto.

Ativando-se variáveis dentro do HOME I/O e conectando-as até o CONNECT I/O é possível atribuí-las a registradores do servidor Modbus TCP. O sistema proposto irá funcionar da seguinte maneira:

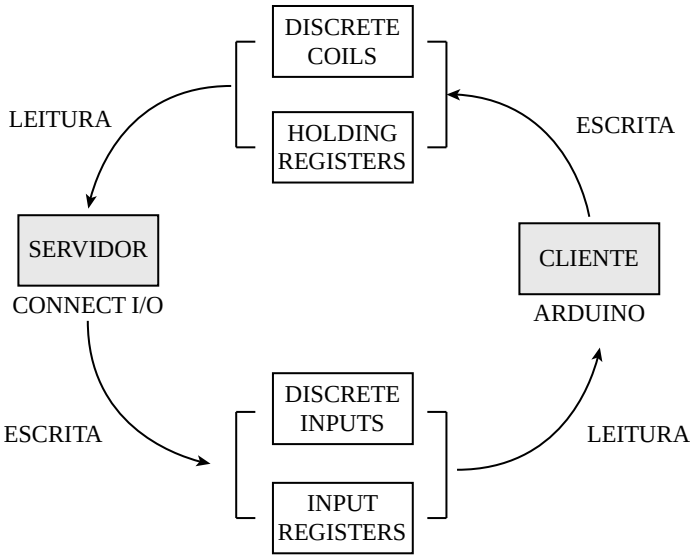

Figura 3 - Esquemático de leitura e escrita entre o servidor e cliente

O sistema deve sempre ler/escrever em variáveis distintas por causa que o CONNECT I/O impões essa restrição no desenvolvimento. Em determinados casos, seria mais fácil simplesmente alterar diretamente o valor de determinada variável.

O diagrama das variáveis atribuídas no CONNECT I/O ficou com a seguinte forma:

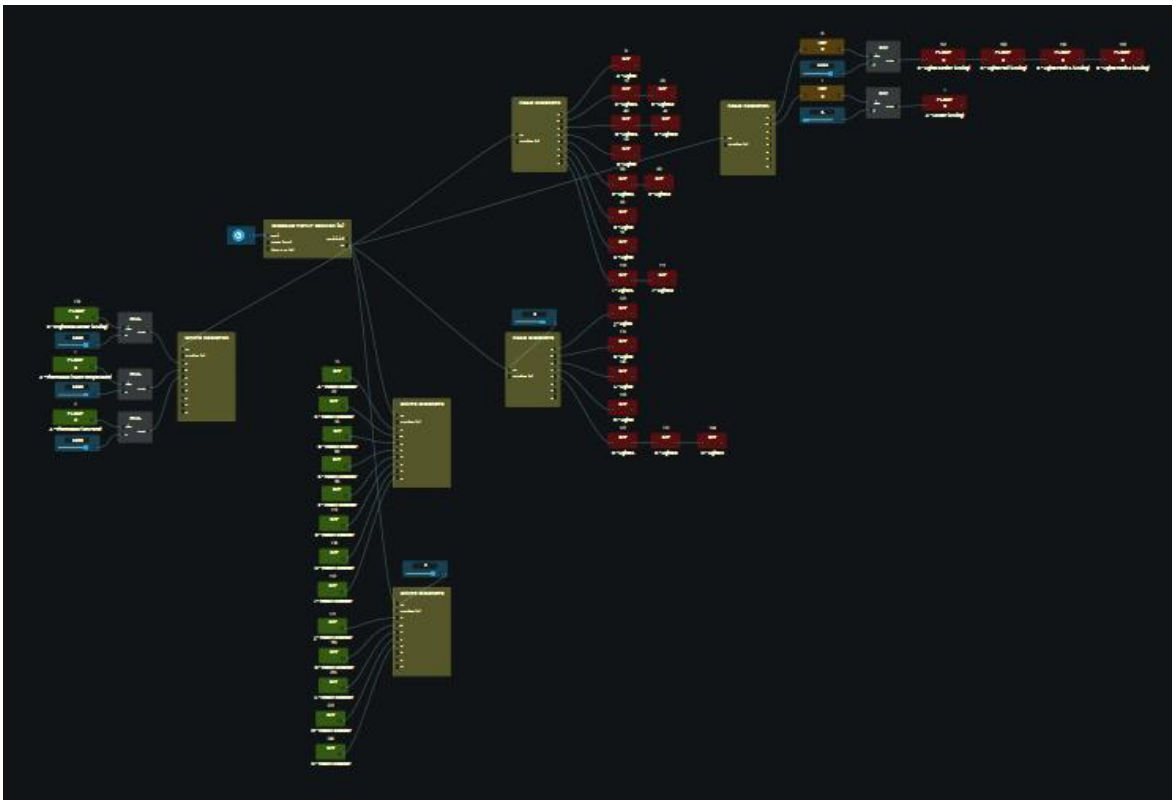

Figura 4 - Diagrama no CONNECT I/O

CAPÍTULO 5 CÓDIGO IMPLEMENTADO

O código escrito implementa as funções descritas para leitura e escrita das variáveis. Também um controle de luminosidade externa, além de um controle PI na temperatura em uma das salas da casa.

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

byte mac[] =      { 0x90, 0xA2, 0xDA, 0x0D, 0xCB, 0xC4 };
byte ip[] =       { 192, 168, 0, 68 };
byte gateway[] =  { 192, 168, 0, 1 };
byte subnet[] =   { 255, 255, 255, 0 };
byte server[] =   { 192, 168, 0, 13 };

int atual,ref,acao,dif;

float ki = 1/100;
float kp = 1/100;
float integralA, proporcionalA, acA;
int temperaturaA, setpointA;


EthernetClient client;

int data;

void MbWriteReg(char ID,int RegAddr, int data)
{
  char buf[12] =
{0x00,0x02,0x00,0x00,0x00,0x06,ID,0x06,RegAddr>>8,RegAddr&0x00FF,data>>8,data&0
x00FF};
  client.write(buf,12);
  while(!client.available());
  while(client.available())
    char c = client.read();
}

void MbWriteCoil_on(char ID,int RegAddr)
{
  char buf[12] =
{0x00,0x02,0x00,0x00,0x00,0x06,ID,0x05,RegAddr>>8,RegAddr&0x00FF,0xFF,0x00};
  client.write(buf,12);
  while(!client.available());
  while(client.available())
    char c = client.read();
}
```

```
void MbWriteCoil_off(char ID,int RegAddr)
{
  char buf[12] =
{0x00,0x02,0x00,0x00,0x00,0x06,ID,0x05,RegAddr>>8,RegAddr&0x00FF,0x00,0x00};
  client.write(buf,12);
  while(!client.available());
  while(client.available())
    char c = client.read();
}

void MbReadInput(char ID,int RegAddr,int *data)
{
  char buf[12] =
{0x00,0x02,0x00,0x00,0x00,0x06,ID,0x04,RegAddr>>8,RegAddr&0x00FF,0x00,0x01};
  client.write(buf,12);

  char resp[11];
  int i=0;
  while(!client.available());
  while(client.available())
  {
    resp[i] = client.read();
    i++;
  }
  *data = resp[9]<<8 | resp[10]&0xFF;
}

void MbReadCoil(char ID,int RegAddr,int *data)
{
  char buf[12] =
{0x00,0x02,0x00,0x00,0x00,0x06,ID,0x02,RegAddr>>8,RegAddr&0x00FF,0x00,0x01};
  client.write(buf,12);

  char resp[11];
  int i=0;
  while(!client.available());
  while(client.available())
  {
    resp[i] = client.read();
    i++;
  }
  *data = resp[9];
}

void setup()
{
  Ethernet.begin(mac, ip, gateway, subnet);
  Serial.begin(9600);
```

```
  delay(1000);

  Serial.println("connecting...");
  while (!client.connect(server, 502));
  Serial.println("connected");

  integralA = 0;
}

void loop()
{
  int valor;

  // Controle de Luminosidade exterior
  int ref = 8000;
  MbReadInput(1,0, &valor); // Leitura da Luminosidade
  valor = valor;
  int erro;
  erro = ref - valor;
  MbWriteReg(1,0, erro); // Escreve direto na luminosidade exterior

    // Lâmpada A
    MbReadCoil(1,0,&data);
    if (data == 1){
       MbWriteCoil_on(1,0);
    } else {
        MbWriteCoil_off(1,0);
      }

    // Temperatura A
    MbReadInput(1, 1, &temperaturaA);
    MbReadInput(1, 2, &setpointA);
    float erroA;
    erroA = (setpointA - temperaturaA)/2000;
    integralA = integralA + erroA;
    acA = integralA;
    if(acA<10 && acA>0){
      acA = acA;
      } else if (acA>10){
        acA = 10;
        } else if (acA<0){
          acA = 0;
        }
    Serial.println(acA);
    MbWriteReg(1,1, acA);


  // Lampada B
    MbReadCoil(1,1,&data);
  if (data == 1){
```

```
   MbWriteCoil_on(1,1);
} else {
    MbWriteCoil_off(1,1);
  }

// Lâmpada D
   MbReadCoil(1,2,&data);
if (data == 1){
   MbWriteCoil_on(1,2);
} else {
    MbWriteCoil_off(1,2);
  }

// Lâmpada E
   MbReadCoil(1,3,&data);
if (data == 1){
   MbWriteCoil_on(1,3);
} else {
    MbWriteCoil_off(1,3);
  }

// Lâmpada F
   MbReadCoil(1,4,&data);
if (data == 1){
   MbWriteCoil_on(1,4);
} else {
    MbWriteCoil_off(1,4);
  }

// Lâmpada G
   MbReadCoil(1,5,&data);
if (data == 1){
   MbWriteCoil_on(1,5);
} else {
    MbWriteCoil_off(1,5);
  }

 // Lâmpada H
   MbReadCoil(1,6,&data);
 if (data == 1){
   MbWriteCoil_on(1,6);
} else {
    MbWriteCoil_off(1,6);
  }

 // Lâmpada I
   MbReadCoil(1,7,&data);
 if (data == 1){
   MbWriteCoil_on(1,7);
} else {
```

```
    MbWriteCoil_off(1,7);
   }

  // Lâmpada J
  MbReadCoil(1,8,&data);
  if (data == 1){
    MbWriteCoil_on(1,8);
  } else {
    MbWriteCoil_off(1,8);
   }

  // Lâmpada K
  MbReadCoil(1,9,&data);
  if (data == 1){
    MbWriteCoil_on(1,9);
  } else {
    MbWriteCoil_off(1,9);
   }

  // Lâmpada L
  MbReadCoil(1,10,&data);
  if (data == 1){
    MbWriteCoil_on(1,10);
  } else {
    MbWriteCoil_off(1,10);
   }

  // Lâmpada M
  MbReadCoil(1,11,&data);
  if (data == 1){
    MbWriteCoil_on(1,11);
  } else {
    MbWriteCoil_off(1,11);
   }

    // Lâmpada N
  MbReadCoil(1,12,&data);
  if (data == 1){
    MbWriteCoil_on(1,12);
  } else {
    MbWriteCoil_off(1,12);
   }

}
```