Vitor Amadeu Souza

Desenvolvendo projetos de

Monitoramento MODBUS

RS485 no ESP32 programado em Arduino

© 2020 by Cerne Tecnologia e Treinamento Ltda.

© 2020 by Vitor Amadeu Souza

Nenhuma parte desta publicação poderá ser reproduzida sem autorização prévia e escrita de **Cerne Tecnologia e Treinamento Ltda.** Este livro publica nomes comerciais e marcas registradas de produtos pertencentes a diversas companhias. O editor utiliza as marcas somente para fins editoriais e em benefício dos proprietários das marcas, sem nenhuma intenção de atingir seus direitos.

Outubro de 2020

Direitos reservados por:
Cerne Tecnologia e Treinamento Ltda

Produção: *Cerne Tecnologia e Treinamento*
E-mail da Empresa: *cerne@cerne-tec.com.br*
Home Page: *www.cerne-tec.com.br.com.br*
Atendimento ao Consumidor: *sac@cerne-tec.com.br*
Contato com o Autor: *vitor@cerne-tec.com.br*



FEITO NO BRASIL

Dedicatória

Como nos meus outros livros, dedico este livro a minha querida esposa Renata Leal.

"Aqueles que, como eu, foram humildes pioneiros da conquista do ar, pensavam mais em criar novos meios de expansão pacífica entre os povos do que lhes fornecer novas armas."

Alberto Santos Dumont

Cerne Tecnologia

A Cerne Tecnologia tem uma equipe preparada para desenvolvimento de projetos eletrônicos em diversas áreas: Médica, Entretenimento, Industrial, Robótica, Científica, Automobilística, Aeronáutica, etc. Trabalhamos com tecnologia microcontrolada usando o PIC, ARM, AVR, 8051, dsPIC, PIC24, PIC32 além do Arduino, Raspberry, Beaglebone etc. Desenvolvemos o projeto desde sua concepção até a entrega do produto final, passando pelas etapas de esquema elétrico, protótipo e desenvolvimento de circuito impresso.

Desenvolvemos aplicativos para smartphones/tablets Android, iOS, Blackberry, Windows Phone e no desenvolvimento de softwares a nível PC para plataforma Windows, usando ferramentas como o Visual Basic, C# e C++.

Atuamos na parte de montagem de placas, onde podemos fornecer ambos os serviços de desenvolvimento de projetos e produção ou apenas um destes.

Desenvolvemos esquemas elétricos e layout de PCI, tanto em tecnologia convencional como SMD.

Temos a flexibilidade de customizar um de nossos produtos, de modo a atender a uma necessidade específica do cliente, tornando o custo de desenvolvimento menor se comparado a construção de um projeto desde a sua fase inicial.

Desenvolvemos e fornecemos kits didáticos para diversos microcontroladores além de apostilas, livros e e-books.

Na hora de desenvolver um projeto ou equipar seu laboratório não hesite em nos contatar. Entre em contato conosco através do endereço cerne-tec.com.br para obter mais informações.



cerne-tec.com.br

Sumário

I. Metodologia de desenvolvimento	**6**
II. MODBUS SLAVE	**7**
III. Medindo a concentração de Amônia	**43**
IV. Amperímetro	**59**
V. Barômetro	**72**
VI. Medindo a concentração de Benzeno	**82**
VII. Célula de Carga	**97**
VIII. Medição de Monóxido de Carbono	**111**
IX. Medição de Dióxido de Carbono	**129**
X. Decibelímetro	**147**
XI. Dinamômetro	**164**
XII. Medindo Etanol em ppm	**174**
XIII. Medindo o fluxo de massa	**187**
XIV. Gaussímetro	**200**
XV. Medindo a concentração de GLP	**215**
XVI. Medindo a concentração de H_2	**231**
XVII. Higrômetro	**246**
XVIII. Medindo a concentração de Ozônio	**256**
XIX. Medição de pH	**271**
XX. Termômetro	**286**
XXI. Medindo a radiação ultravioleta	**296**
XXII. Voltímetro	**312**

Capítulo I

Metodologia de desenvolvimento

1. Introdução

Esta literatura é uma continuação da obra *Programando o ESP32 no Arduino (2018)* do mesmo autor e editora, onde a proposta é desenvolver um sensor medidor seguindo o sumário da obra que forneça o resultado de tal medição através da rede MODBUS RS485 programado no Arduino.

A placa didática utilizada foi a ESP32 wroom, onde tal kit está à venda no site www.cerne-tec.com.br.

Capítulo II

MODBUS SLAVE

1. Introdução

O protocolo MODBUS é um dos mais usados a nível industrial atualmente. O mesmo é usado por CLPs e Computadores para monitorarem, por exemplo, uma rede de periféricos como sensores, válvulas, entradas analógicas e saídas analógicas através desta rede. Veja abaixo algumas características deste protocolo:

- Utilizado para comunicação entre CLPs;
- Define a estrutura de quadros, não de meio físico;
- Usualmente utilizado sobre RS232, RS485 ou Ethernet;
- Arquitetura do tipo Mestre/Escravo;
- Permite Broadcast;
- Apenas dois tipos de quadros.

O protocolo MODBUS foi desenvolvido pela Modicon Industrial Automation Systems, hoje Schneider, para comunicar um dispositivo mestre com outros dispositivos escravos. Embora seja utilizado

normalmente sobre conexões seriais padrão RS-232, ele também pode ser usado como um protocolo da camada de aplicação de redes industriais tais como TCP/IP sobre Ethernet e MAP. Este é talvez o protocolo de mais larga utilização em automação industrial, pela sua simplicidade e facilidade de implementação.

2. Tipos de quadros

Os tipos de quadros ou frames disponíveis para o MODBUS são os seguintes:

- Quadro de consulta;
- Endereço;
- Código da função;
- Dados;
- Verificação de erro.

Ou seja, o quadro que pode fazer a consulta, por exemplo, em um sensor ou CLP contém nele um campo que identifica o endereço no qual o mestre deseja se comunicar, neste caso com o escravo podendo ser do endereço de 1 a 247. Caso o endereço seja 0, é

identificado como um tipo de mensagem broadcast, onde todos os elementos da rede podem receber tal dado. Em seguida, é informado o código da função que identifica o que o frame em si faz. Seguido do identificador da função vem a parte de dados, onde neste há o pacote de dados referente as informações que serão enviadas. Finalmente há a parte de verificação de erros, onde nesta parte há checagem de dados enviados, a fim de verificar se foi enviado corretamente. O próximo frame é o de resposta, composto dos seguintes itens:

- Quadro de resposta;
- Endereço;
- Confirmação (Eco do código de função enviado);
- Dados;
- Verificação de erro.

3. Modos de Transmissão

Existem dois tipos de modos de transmissão usados no MODBUS. O primeiro deles chama-se modo do tipo ASCII e outro do tipo RTU. No mercado industrial e em geral, o modo do tipo RTU é o mais usado, sendo menos o do tipo ASCII. No modo de transmissão

ASCII há o desperdício de dados do canal, pois cada caracter enviado pela rede é enviado como um caracter ASCII enquanto que no modo RTU, o dado é transmitido em forma de bits. Observe abaixo como é formado o frame do tipo RTU:

Start	Endereço	Função	Dados	CRC	END
Silêncio 3..5 chars	← 8 bits→	← 8 bits→	←N x 8 bits→	←16 bits→	Silêncio 3..5 chars

Observe primeiramente que temos um período de silêncio, onde sempre que as linhas de comunicação ficarem sem tráfego de dados pelo tempo de até 3,5 caracteres, o escravo pode entender que a rede entrou em estado idle, ou seja, de repouso. Como a comunicação utilizada terá uma taxa de comunicação de 9600 bps, pode observar que o tempo de transmissão de 1 bit será de :

$$1/9600 = 104\mu s$$

Como a comunicação utilizada terá 1 bit de start, 8 de dados e 1 de stop, ao todo temos 10 bits. Para achar o tempo para transmitir 1 byte na rede MODBUS, multiplique 104 x 10, achando neste caso o tempo de 1,04 ms. Ou seja, o tempo para transmitir 1 byte no MODBUS são de 1,04 ms e caso a rede fique 3,64 ms em inatividade

(que é referente a 3,5 caracteres), tem-se neste caso o estado de silêncio ou estado idle. Em seguida encontramos os 8 bits referentes a endereço. Note que caso este byte esteja em 0, indica que a comunicação é do tipo broadcast, ou seja o mestre comunicará com todos os elementos da rede e caso o endereço seja de 1 a 247 indica que comunicará com um elemento específico da rede. No campo de função temos a indicação do que se trata o comando que está sendo enviado. Neste caso, esta tabela segue a tabela abaixo:

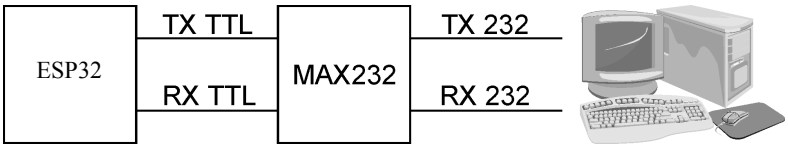
Código	Descrição
01	Lê um número de variável de saídas digitais
02	Lê um número de variável de entradas digitais
03	Lê um número de variável de registros retentivos
04	Lê um número de variável de entradas analógicas
05	Altera o estado de uma saída digital
06	Altera o estado de uma saída analógica
07	Lê exceções
08	Funções de diagnóstico
15	Força uma quantidade variável de saídas
16	Preset de uma quantidade variável

No campo de dados estão os dados referentes ao tipo de comando enviado. Finalmente o campo de CRC que é a checagem de dados.

4. Comunicação RS232

Antes de entrar na comunicação propriamente dita MODBUS, é importante abordar a comunicação RS232. Todos os exemplos propostos neste livro estão escritos no Arduino IDE. A comunicação utilizada é do tipo RS232. A taxa de comunicação será de 9600 bps, sem paridade com 1 stop-bit e um start-bit.

Os níveis de comunicação usados pelo RS232 são diferentes do nível utilizado pelo ESP32. Neste caso, normalmente é utilizado um CI chamado MAX232 que converte de TTL (é o nível do ESP32) para RS232 (é o nível da entrada de dados do PC). Veja um diagrama de blocos para facilitar a visualização:



A função do MAX232 nesse circuito é simplesmente converter o nível TTL no padrão 232 e vice-versa, já que ele também converte 232 em TTL. A comunicação do tipo RS232 é chamada de full-duplex, pois é uma linha somente de transmissão e outra somente de recepção. Isso quer dizer que enquanto o sistema está transmitindo um byte pela linha de TX, ele pode perfeitamente receber outro pela

linha de RX. Observe os níveis associados aos padrões TTL e receber RS232.

Nível	TTL	RS232
1	5V	-3V até -15V
0	0V	3V até 15V

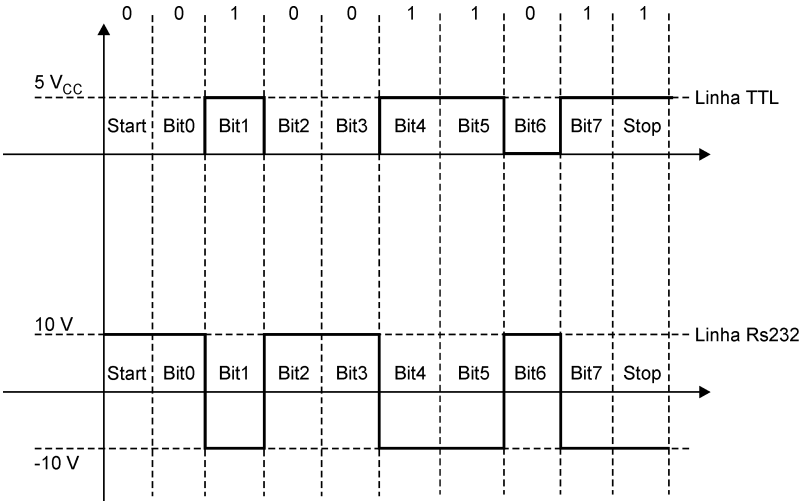
Verifique que o nível 1 no padrão TTL está associado ao 5 V_{CC} enquanto no RS232 está associado à tensão de –3 até –15V. Qualquer tensão nessa faixa (-3 até -15V) será entendida como 1 no receptor do sistema. Já o 0 lógico está associado 0VCC no padrão TTL enquanto no RS232 está associado de 3 até 15V.

Para descobrir o tempo de transmissão de 1 bit, basta com que se divida 1 pela taxa de comunicação (baud-rate). Desta forma, encontra-se o período ou tempo de 1 bit na comunicação. Neste caso, que está sendo utilizada uma taxa de 9600bps, o tempo do bit é:

$$T= 1 / 9600 = 104\ \mu s \text{ (aproximadamente)}$$

Toda comunicação serial começa com um bit de início que é chamado de start-bit. Logo em seguida os bits de dados começando pelo bit menos significativo do byte a ser transmitido. Opcionalmente, pode ser usada a paridade. Como neste caso este

item não está incluído, a comunicação acaba com um bit de parada ou *stop-bit.* Como exemplo, vamos enviar o byte 10110010 para o PC a uma taxa de 9600bps sem paridade e 1 stop-bit. Como já visto através do cálculo anterior, o tempo de 1 bit é de aproximadamente de 104us. O gráfico é plotado na linha TTL e também na linha RS232. Veja:



Verifique que o bit de start sempre começa em 0 seguido dos bits de dados e finalizado pelo bit de stop que sempre finaliza em 1, e a linha de dados volta a seu estado de repouso.

Este exemplo usa uma linha de transmissão e recepção que possibilitam a comunicação serial. Observe que a placa ESP32 wroom

possui na placa um conversor TTL<->USB usando o circuito integrado CH340.

Deve-se configurar a velocidade no qual é feita a comunicação, podendo esta ser de várias velocidades como 1200 bps (bits por segundo), 2400 bps, 4800 bps, 9600 bps etc. Neste exemplo a velocidade usada é de 9600 bps. Abaixo está apresentado a função que configura a velocidade de comunicação:

```
Serial.begin(velocidade);
```

A função para enviar a string está apresentada a seguir:

```
Serial.println(texto);
```

O exemplo completo está apresentado logo abaixo enviando a string para o PC via RS232 a 9600 bps.

```
void setup()
{
  Serial.begin(9600);
  Serial.println("Aprendendo Arduino");
}

void loop()
{

}
```

Para visualizar os dados será utilizado o Serial Monitor, que já vem com o próprio software Arduino. Clique na opção marcada abaixo para inicializar este software.



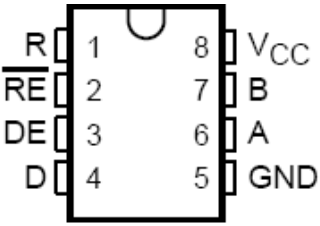
Neste momento, a seguinte tela surgirá, onde após a gravação da placa e remoção do jumper de gravação será possível visualizar o resultado apresentado abaixo a 9600 bps.



5. Comunicação RS485

O RS485 é uma evolução do padrão RS422, tendo como principal enfoque a comunicação em rede, ou seja, com apenas um par de fios é possível se comunicar com diversos equipamentos em rede usando o mesmo barramento. Assim como o RS422, o RS485 utiliza linha de dados balanceada, bastante similar as linhas de dados da interface RS422, logo também permite a comunicação em distâncias de até 1200 metros de maneira extremamente confiável.

A interface de hardware para comunicação RS485 utilizada é baseada no chip 75176, um conversor TTL para RS485, observe abaixo a pinagem do mesmo:



Veja as funções destes pinos na tabela abaixo:

Pino	Função
1	Saída de dados TTL
2	Seleção de modo TX ou RX
3	Seleção de modo TX ou RX
4	Entrada de dados para transmissão
5	GND
6	Saída de dados diferencial A
7	Saída de dados diferencial B
8	VCC

Os níveis elétricos 1 e 0 na interface RS485:

Function Tables

DRIVER

INPUT D	ENABLE DE	OUTPUTS A	OUTPUTS B
H	H	H	L
L	H	L	H
X	L	Z	Z

RECEIVER

DIFFERENTIAL INPUTS A – B	ENABLE $\overline{RE}$	OUTPUT R
$V_{ID} \geq 0.2$ V	L	H
-0.2 V $< V_{ID} < 0.2$ V	L	?
$V_{ID} \leq -0.2$ V	L	L
X	H	Z
Open	L	?

H = high level, L = low level, ? = indeterminate, X = irrelevant, Z = high impedance (off)

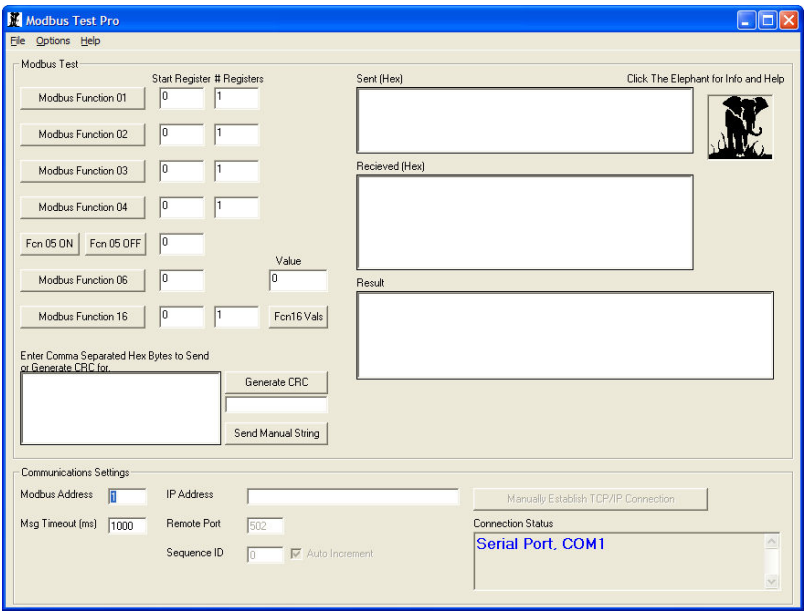
Conecte o pino TX/RX do 75176 aos pinos de comunicação serial do ESP32 além dos pinos de controle 2 e 3 ao pino 3.

O leitor deve possuir um conversor USB para RS485 para estabelecer a comunicação, como sugere a figura abaixo.

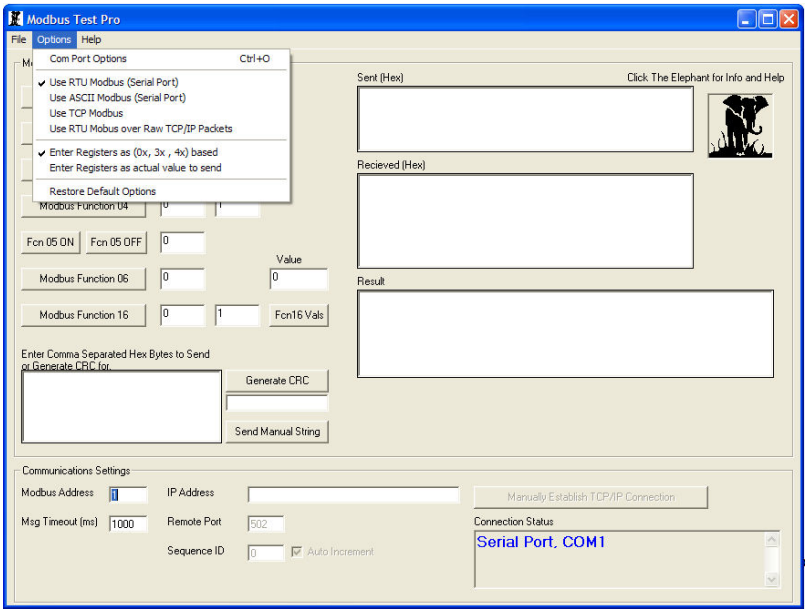


6. Software de Comunicação MODBUS

Existem vários softwares que permitem a comunicação com dispositivos MODBUS. Neste livro, será usado o software MODBUS Test Pro, que pode ser visualizado abaixo:



Este software pode funcionar em modo serial ou TCP/IP, bastando para isso selecionar o modo indo no menu Options, neste momento a seguinte tela surgirá:

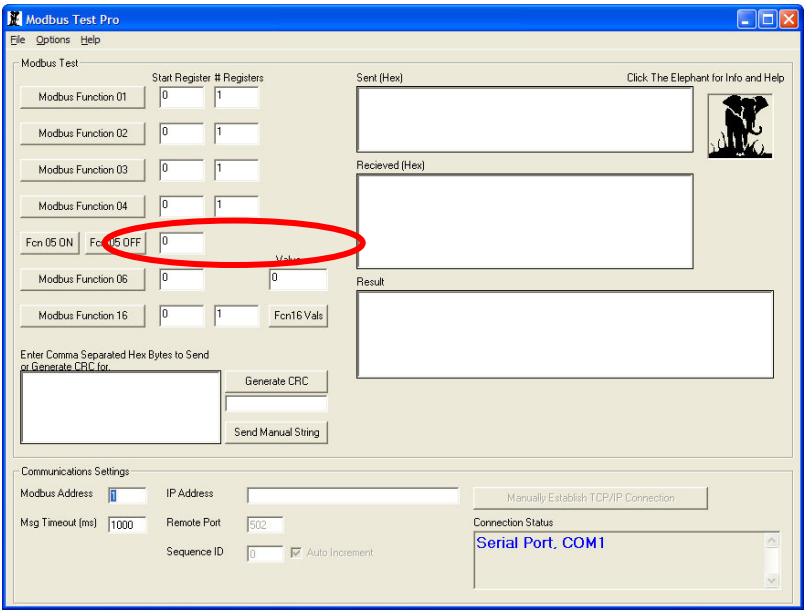


Observe que além de poder escolher o funcionamento entre Serial e TCP/IP, podemos escolher a opção RTU assim como ASCII. Deixe neste caso a opção *Use RTU MODBUS (Serial Port)* marcada. Você poderá baixar este software gratuitamente no seguinte link: www.rogue-engr.com.

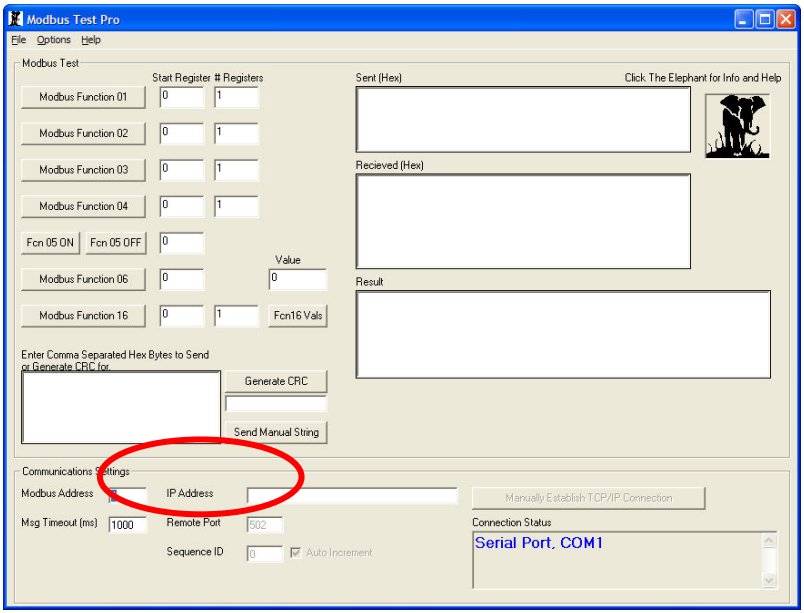
Teste melhor as funcionalidades deste software, observando melhor o seu funcionamento, pois em breve usaremos este software com o processador.

7. Protocolo de Comunicação

No exemplo proposto no livro, o objetivo é ligar e desligar um led presente na placa didática usando o protocolo MODBUS. Para isso é utilizada a interface RS485 e o endereço da placa será o endereço 1. Será usado o comando 5 da tabela de comandos do MODBUS neste exemplo, que permite ligar ou desligar uma saída na rede MODBUS. O mesmo está marcado abaixo, observe:

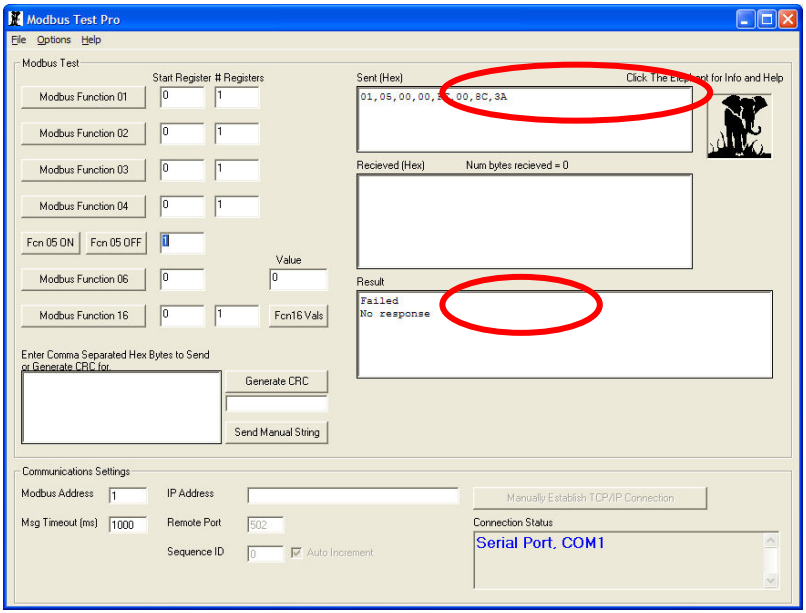


Especifique o endereço do dispositivo que deseja acionar na caixa apresentada abaixo:



Neste caso informe o endereço 1 que será o endereço programado no processador.

Pressione o botão Fcn 05 ON, o resultado será o seguinte:

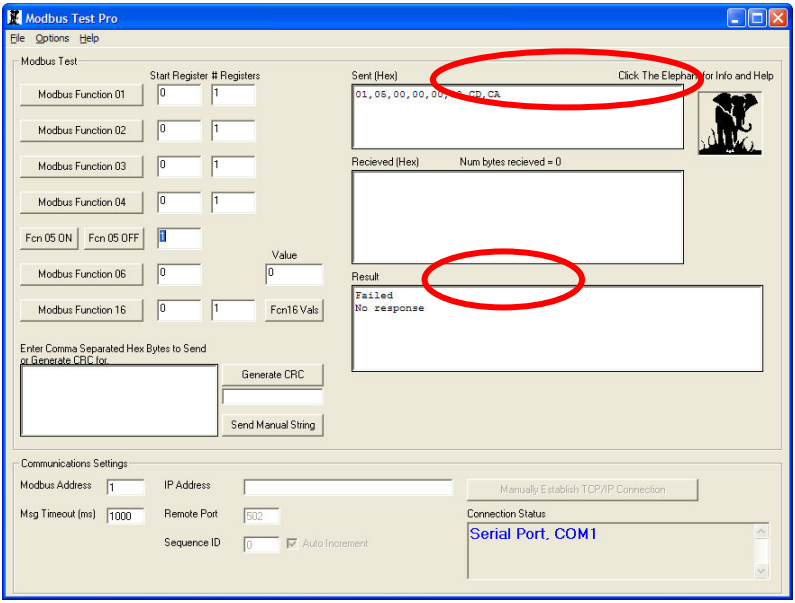


No campo Send (Hex) é apresentado o conjunto de dados enviado. Observe que neste caso foram enviados os seguintes dados: 01, 05, 00, 00, FF, 00, 8C, 3A. Agora observe estes dados enviados com o protocolo apresentado abaixo do quadro de funcionamento RTU:

Start	Endereço	Função	Dados	CRC	END
Silêncio 3..5 chars	← 8 bits→	← 8 bits→	←N x 8 bits→	←16 bits→	Silêncio 3..5 chars

O endereço que foi enviado é 0x01, conforme configurado no campo apresentado. Em seguida, o valor 0x05 refere-se à função

usada que é a função 5 que permite o acionamento ou não de saídas digitais (bobinas). Em seguida o valor 0x00, 0x00, 0xFF e 0x00 são referentes aos dados, onde neste caso representa o acionamento de uma saída na placa didática conectada ao MODBUS. Finalmente o campo 0x8C e 0x3A são referentes ao campo de checagem de erro. O campo Result ficou em *Failed, No response,* pois o microcontrolador não está conectado ao PC. Pressione o botão Fcn 05 OFF, o resultado será o seguinte:



Os dados enviados foram: 0x01, 0x05, 0x00, 0x00, 0x00, 0x00, 0xCD e 0xCA. Analise estes dados juntamente com o quadro apresentado abaixo:

Start	Endereço	Função	Dados	CRC	END
Silêncio 3..5 chars	← 8 bits→	← 8 bits→	←N x 8 bits→	←16 bits→	Silêncio 3..5 chars

Novamente o valor 0x01 é referente ao endereço e 0x05 ao comando usado que neste caso é de acionamento de saídas. Os valores 0x00, 0x00, 0x00 e 0x00 são referentes aos dados onde neste caso é utilizado para desligar uma saída e 0xCD e 0xCA referentes ao resultado da checagem de erros.

Neste tipo de comando caso algum dispositivo receba corretamente este dado o mesmo deve retornar com os mesmos comandos, pois assim o controlador (CLP ou Computador) saberá que o dispositivo executou corretamente este comando.

A técnica apresentada acima é válida para todos os outros comandos do MODBUS.

8. Programa para o Microcontrolador

Abaixo o programa que irá rodar no microcontrolador para funcionar juntamente com o programa apresentado acima. Este programa irá receber 8 bytes enviados pela serial e em seguida verifica se no byte 0 foi recebido o endereço correto, neste caso o endereço 0x01 que foi o endereço configurado na placa.

Em seguida, verifica se o comando recebido é o 0x05 referente ao acionamento ou não de uma saída digital. Caso seja, verifica se o dado enviado foi 0xFF ou 0x00. Caso tenha sido o 0xFF o rele da placa é acionado imediatamente e caso não seja o rele é desligado.

Observe abaixo a listagem do programa que foi feito no Arduino e a comunicação usada é do tipo RS485, porém nada impede que seja usada a RS232, por exemplo.

Veja que também há no loop principal do programa uma variável chamada timeout que verifica constantemente se ocorreu o estado idle no barramento e caso tenha ocorrido isso verificando se esta chegou ao valor de 3640 us ou 3,64 ms (conforme comentado no início do livro) reinicia a variável de índice que é responsável por receber os dados da serial e armazenar os dados no buffer da serial.

```
/*  *  *  *  *  *  *  *  *  *  *  *  *
 *          Cerne Tecnologia          *
 *        www.cerne-tec.com.br        *
 *    Projetos em Arduino - MODBUS    *
 *  *  *  *  *  *  *  *  *  *  *  *  */

/*  *  *  *  *  *  *  *  *  *  *  *  *
 *       Declaração de constantes     *
 *  *  *  *  *  *  *  *  *  *  *  *  */
#define  ENDERECO  1

/*  *  *  *  *  *  *  *  *  *  *  *  *
 *       Declaração de variáveis      *
 *  *  *  *  *  *  *  *  *  *  *  *  */
int timeout=0;
char indice=0;
char buffer[20];

/*  *  *  *  *  *  *  *  *  *  *  *  *
 *          Declaração de I/Os        *
 *  *  *  *  *  *  *  *  *  *  *  *  */
#define  LED       2
#define  CTR       3

/*  *  *  *  *  *  *  *  *  *  *  *  *
 *        Função de inicialização     *
 *  *  *  *  *  *  *  *  *  *  *  *  */
void setup(void)
{
  pinMode(LED,OUTPUT);
  pinMode(CTR,OUTPUT);
  digitalWrite(CTR,0);
  Serial.begin(9600);
}
```

```
/*  *  *  *  *  *  *  *  *  *  *  *  *
 *          Função de loop            *
 *  *  *  *  *  *  *  *  *  *  *  *  */
void loop(void)
{
  timeout++;
  delayMicroseconds(1);
  if(timeout==3640)
  {
     timeout=0;
     indice=0;
  }

  if(Serial.available()>0)
  {
    buffer[indice]=Serial.read();
    indice++;
    timeout=0;

    if(indice==8)
    {
      indice=0;
      if(buffer[0]==ENDERECO)
      {
        if(buffer[1]==5)
        {
            digitalWrite(CTR,1);delay(10);

            Serial.write(buffer[0]);
            Serial.write(buffer[1]);
            Serial.write(buffer[2]);
            Serial.write(buffer[3]);
            Serial.write(buffer[4]);
            Serial.write(buffer[5]);
            Serial.write(buffer[6]);
            Serial.write(buffer[7]);

            delay(10);digitalWrite(CTR,0);

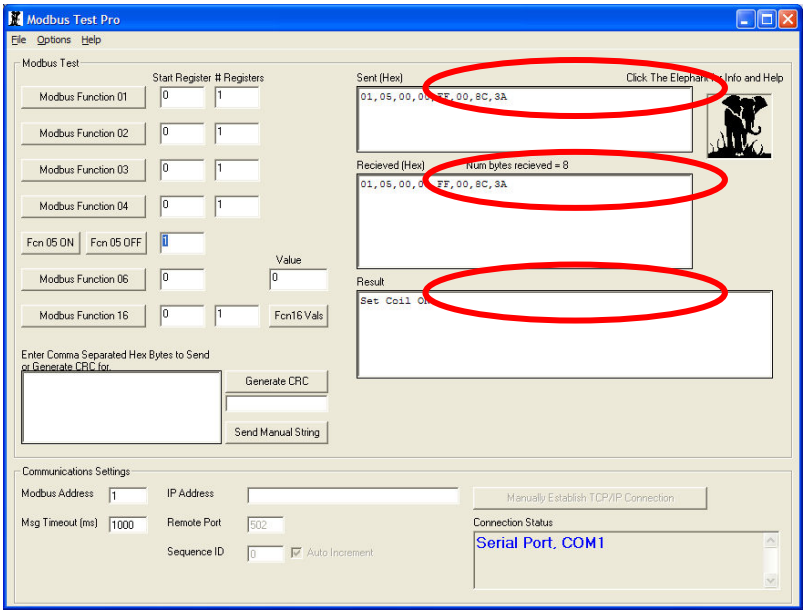
            if(buffer[4]==0xFF)
```

```
                digitalWrite(LED,1);

              if(buffer[4]==0x00)
                digitalWrite(LED,0);

          }
        }
      }
    }
}
```

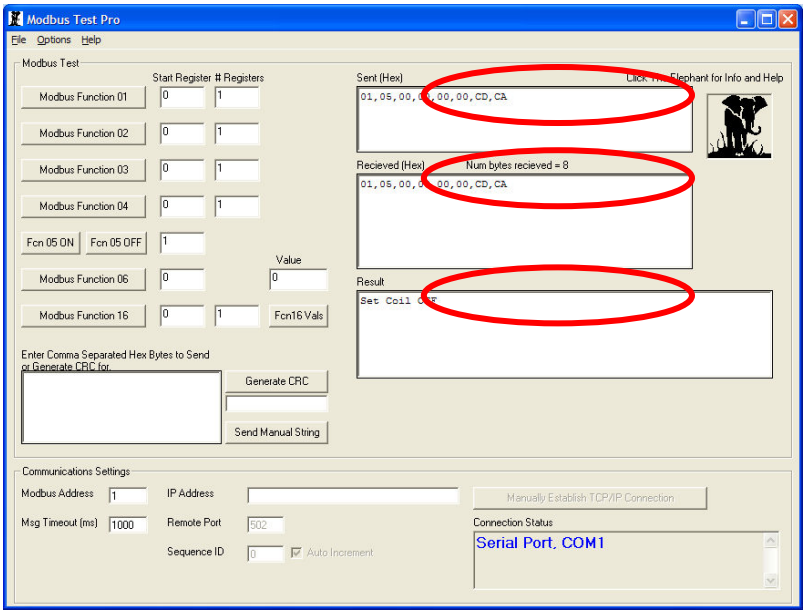
Observe a resposta da placa enviando o comando para ligar o rele com a placa devidamente programada e ligada na porta serial do PC:



Os comandos enviados são recebidos novamente da mesma forma que foram enviados, assim o controlador sabe que o device (dispositivo ou escravo MODBUS) recebeu os dados corretamente.

No campo Result veja que o dado foi corretamente recebido, pois o resultado é *Set Coil ON.*

Clicando o botão Fcn 05 OFF o resultado será o seguinte:



Note que neste caso o frame enviado foi diferente já que o campo de dados está todo em 0 e o resultado foi Set Coil OFF ou seja a saída desligada. Neste momento a saída será desligada.

9. Calculando o CRC

É um projeto em que será retornado para o controlador que neste caso é o PC o valor de um botão presente na Placa Cerne RS485, ou seja, pressionado o valor retornado será 0x01 e caso esteja solto o valor retornado será 0x00. Neste caso no momento do retorno do dado deve-se calcular o CRC da mensagem. Veja a rotina aplicada neste caso apresentada abaixo:

```
void calc_crc(unsigned int dataLength)
{
   unsigned int CheckSum;
   unsigned int j;
   unsigned char lowCRC;
   unsigned char highCRC;
   unsigned char i;
   CheckSum = 0xffff;

    for (j=0; j<dataLength; j++)
    {
       CheckSum = CheckSum^(unsigned int)buffer[j];
       for(i=8;i>0;i--)
          if((CheckSum)&0x0001)
             CheckSum = (CheckSum>>1)^0xa001;
          else
             CheckSum>>=1;
    }

    highCRC = CheckSum>>8;
    CheckSum<<=8;
    lowCRC = CheckSum>>8;
    buffer[dataLength] = lowCRC;
    buffer[dataLength+1] = highCRC;

}
```