Logbook

MAN4HEALTH

João Paulo Coelho

Sensor de CO2

Fevereiro 2023

1. Esquemático e PCB

Nesta seção mostra-se as diferentes versões do esquemático e PCB associados ao nó para medição de CO2, temperatura e humidade do ar que ficará colocado no poste de transmissão LoRa.

1.1. Versão 1.0

Foram detetados alguns erros nesta primeira versão depois de manufaturada. Em particular, e o pior dos erros, foi a falta de ligação entre o transceiver RS485 com o ATMega328. Uma situação que será corrigida na versão seguinte.

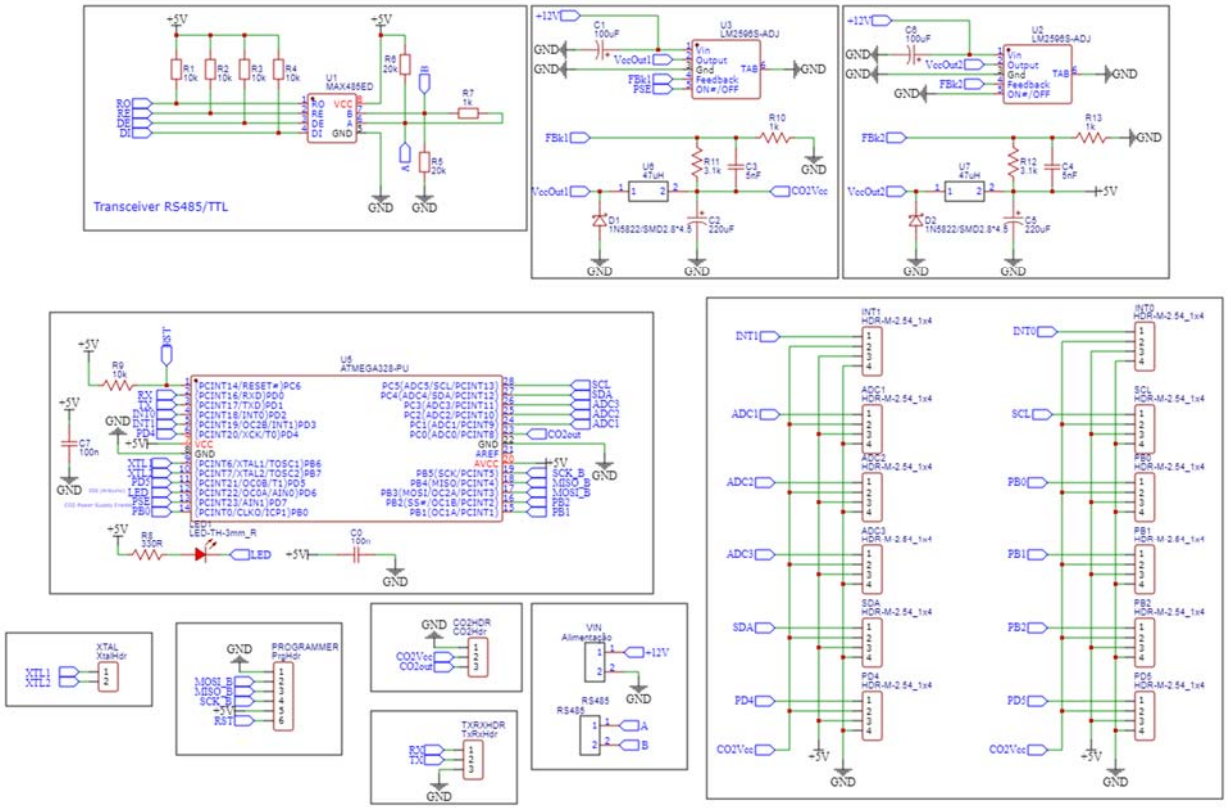


Figura 1 - Primeira versão do esquemático do nó de medida de CO2.

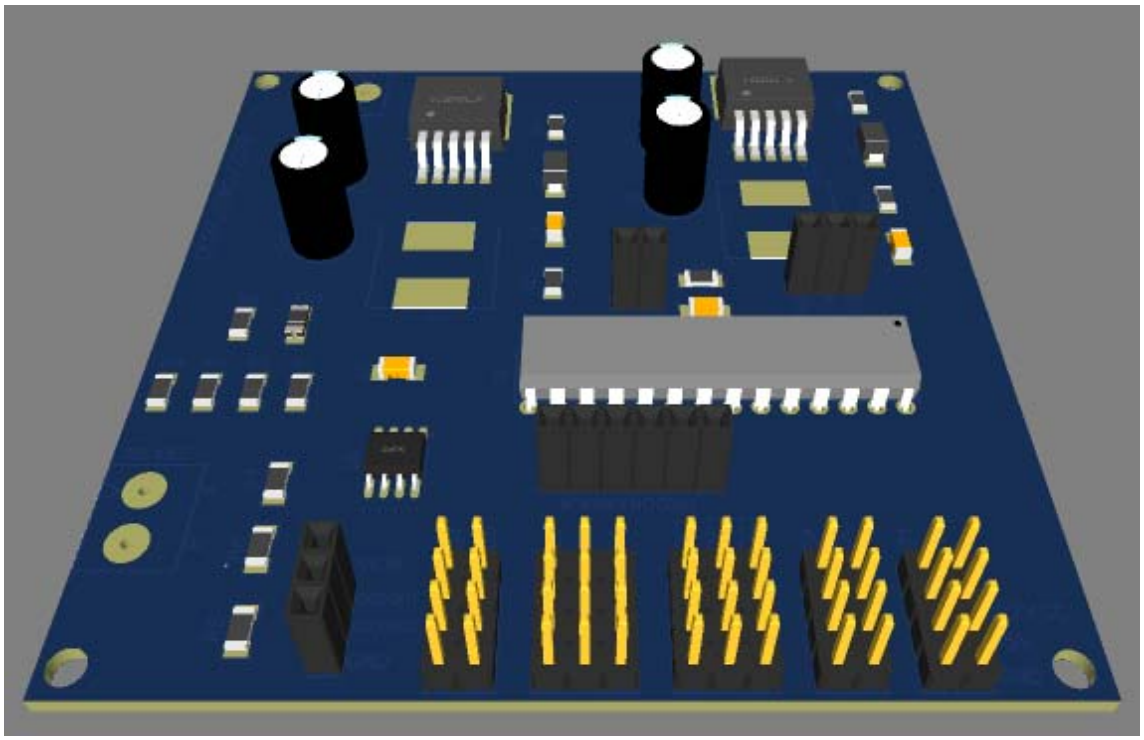
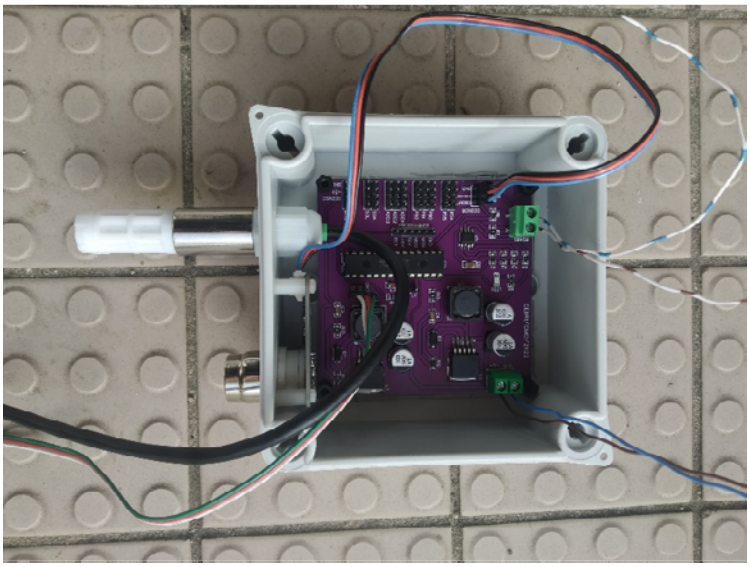


Figura 2 - Placa de circuito impresso relativo à primeira versão do nó de medida de CO2.

Todo o projeto pode ser encontrado na minha conta do EasyEDA. Cópias foram criadas e carregadas no GitHub do projeto.

Primeira versão do hardware

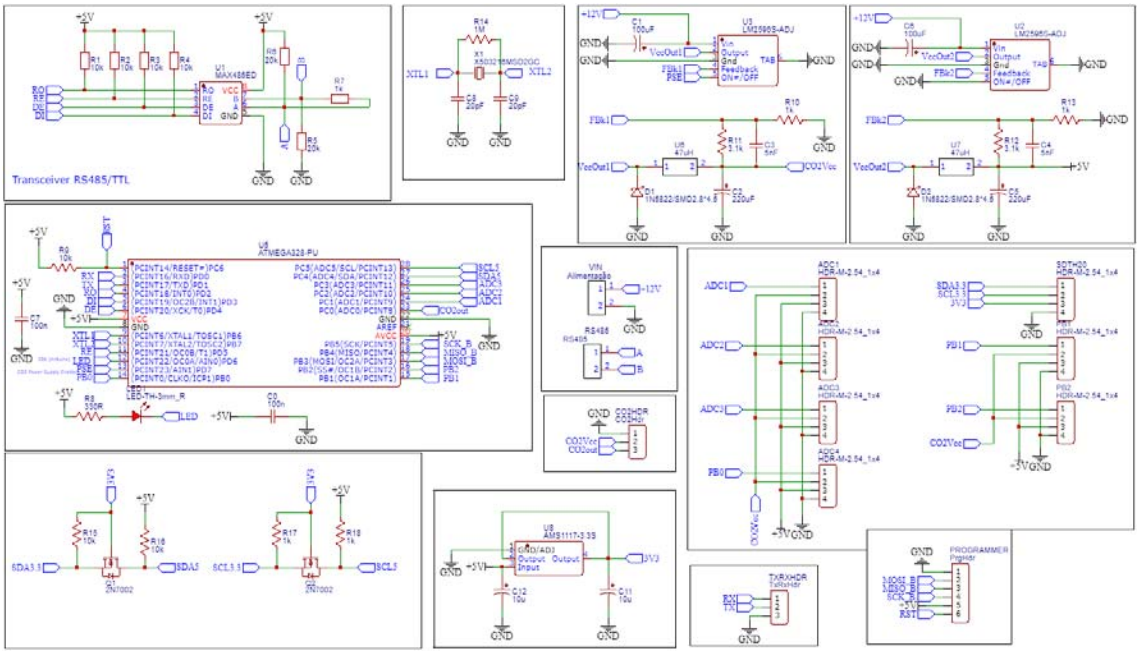


N.B. *Foi incluído um patch em hardware para lidar com a necessidade de se alimentar e comunicar com o sensor de humidade e temperatura usando 3.3V. Esse patch não se mostra nesta fotografia.*

Alterações a fazer

A primeira versão do PCB tem dois problemas. O primeiro resulta de um erro no esquemático que não contempla a ligação das portas RO/RE/DE e DI aos pinos respetivos do transceiver RS485. O segundo, mais subtil, deve-se ao facto deste circuito considerar o uso do clock interno do ATMega328. Este clock interno, com valor igual a 8 MHz é incompatível com uma transmissão série igual a 115200 bps que é o que é utilizado pela estação meteorológica colocada no local. Deste modo, será necessário alterar o PCB de modo a incluir um cristal de 11.0592 MHz. Ambos os problemas foram endereçados temporariamente através de *jumpers* (para o primeiro caso) e adição de um cristal externo no segundo caso. A figura em baixo ilustra esta abordagem.

1.2. Versão 2.0



2. Instalação do MiniCore

A programação deste módulo de medida, considerando um cristal externo com a frequência referida anteriormente, deve ser conduzida tendo em consideração esta alteração. Deste modo, o uso do método descrito para o nó LoRa (usando o breadboard-avr) não é aplicável assim como o AVR boards para placas Arduino nativas que limitam a escolha do oscilador a 16 MHz. Assim, para lidar com este problema, foi instalado um add-on designado por MiniCore que permite programar o ATMega328 com diferentes valores para o cristal. Nomeadamente o valor que é utilizado no presente circuito. O MiniCore adiciona suporte ao Arduino IDE para muitas

configurações da família de microcontroladores AVR, incluindo o ATmega328P. A metodologia usada para instalar este add-on ao Arduino IDE encontra-se documentado a seguir (para a versão 1.8.19 do Arduino IDE):

1. Selecione **Ficheiro > Preferências** nos menus do Arduino IDE.
2. Digitar o seguinte URL no campo "**URL Adicionais do Gestor de Placas**":
   https://mcudude.github.io/MiniCore/package_MCUdude_MiniCore_index.json
   A Figura 3 mostra o procedimento.
3. Selecione **Ferramentas > Placa > Gestor de Placas** nos menus do Arduino IDE e escreva na barra de procura “MiniCore”. Selecione "MiniCore by MCUdude" clicando no botão Instalar.

A instalação do MiniCore adicionou algumas novas opções de placas ao Arduino IDE. Para isso deve selecionar o microcontrolador através do menu **Ferramentas > Placa > MiniCore > ATmega328** como se mostra na Figura 4. Posteriormente, deve aceder-se à opção **Clock** e selecionar **11.0592 MHz**.

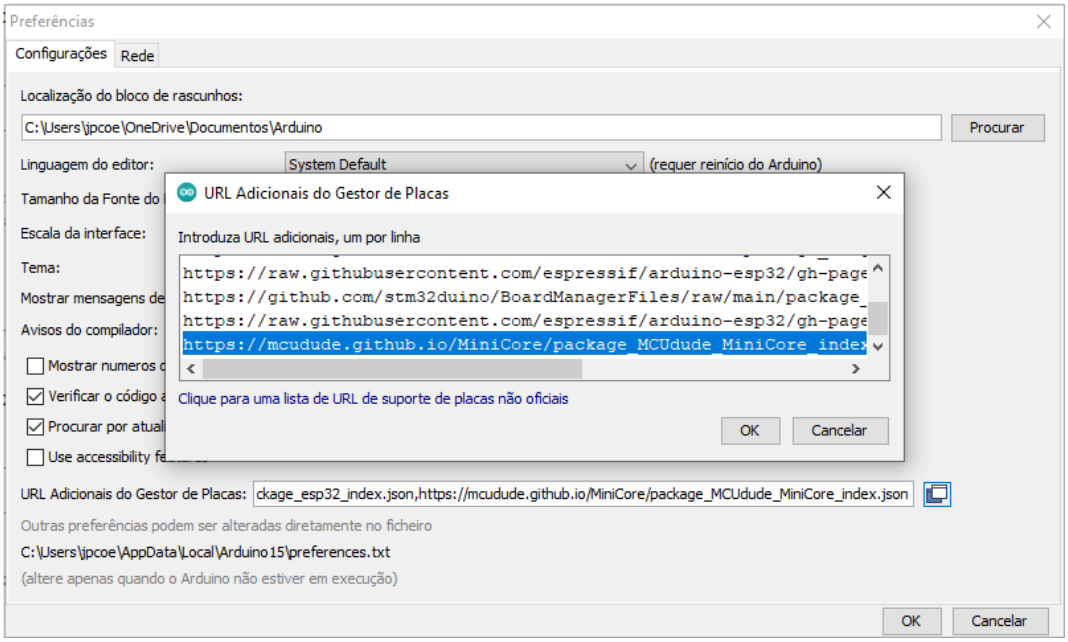


Figura 3 - Janela onde deve ser adicionado o URL para novas placas.

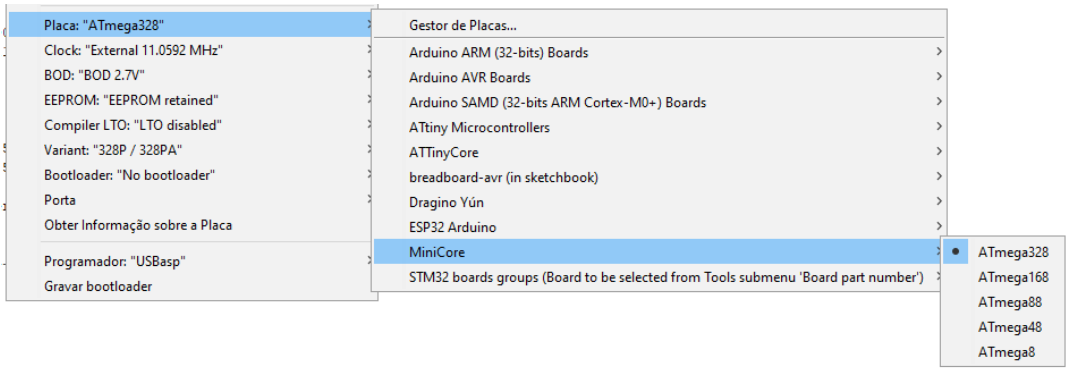


Figura 4 - Seleção do microcontrolador através do gestor do MiniCore.

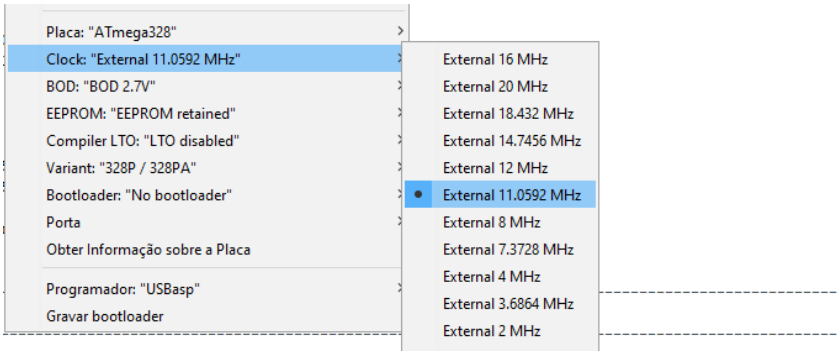


Figura 5 - Opções de clock que podem ser selecionadas.

Todo este procedimento, assim como documentação adicional, pode ser encontrada no repositório:

https://github.com/MCUdude/MiniCore#readme 3

Antes de terminar esta seção, ficam aqui algumas considerações para memória futura. Primeiro, deve ser definido o tipo de programador. No nosso caso, a utilização do USBasp é o método pelo qual se optou para programar o microcontrolador. Adicionalmente, e isto é fundamental, é necessário executar a opção "Gravar bootloader" para configurar os "fuses" para as opções corretas. Obviamente, para realizar esta operação, o microcontrolador deve estar ligado ao programador. A Figura 6 mostra um screenshot do processo.

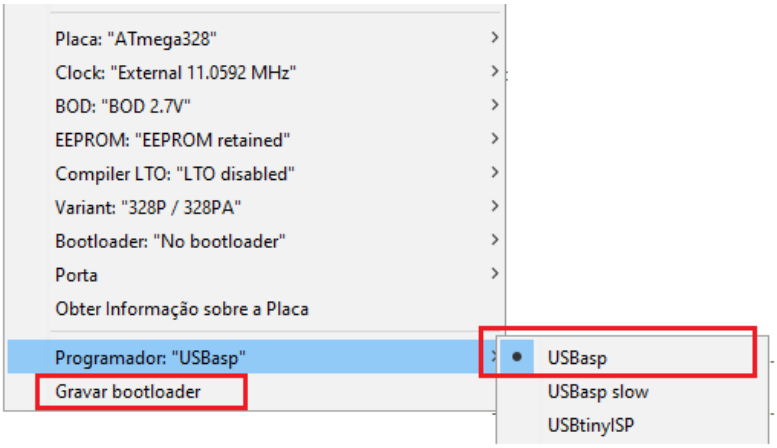


Figura 6 - Parametrização adicional do MiniCore.

3. Ensaios para Depuração do Hardware

No repositório do GitHub do projeto podem ser encontrados diferentes *skeches* desenhados para se avaliar os diferentes vértices do hardware. Em particular, o LED incluído na PCB, a ativação do conversor DC/DC para alimentação dos sensores (em particular do sensor de CO2) e a comunicação via RS232. Alguns desses elementos serão incluídos em *skeches* mais elaborados para testar outras caraterísticas do hardware como é o caso da comunicação Modbus via RS485. É exatamente neste contexto que se escreve esta seção.

3.1. Comunicação RS485

O primeiro ensaio documentado envolve a depuração de toda a cadeia de comunicação via RS485. Para isso, foi realizado o conjunto de ligações apresentado na Figura 7

Por conveniência, o sketch carregado no microcontrolador encontra-se na caixa de texto que se segue, mas também pode ser encontrado no GitHub do projeto:

```
#include <SoftwareSerial.h>
#include <TinyRS485.h>
#include <TinyModbus.h>
#define MAX485_DE 4
#define MAX485_RE 5
#define MAX485_RO 2
#define MAX485_DI 3

SoftwareSerial SoftSerial(MAX485_RO, MAX485_DI); // RX, TX
TinyRS485 RS485;
// ----------------------------------------------------------------------------
// void setup(void)
// ----------------------------------------------------------------------------
void setup()
{
  Serial.begin(115200);     // Porto série (hardware) para debug
  SoftSerial.begin(115200);   // Porto série (software) usado pelo transceiver RS485
  pinMode(MAX485_DE, OUTPUT);
  pinMode(MAX485_RE, OUTPUT);
  digitalWrite(MAX485_DE, LOW); // Desativa DRIVER
  digitalWrite(MAX485_RE, LOW); // Ativa RECEIVER
  RS485.begin(SoftSerial, MAX485_DE, MAX485_RE);
}
// ----------------------------------------------------------------------------
// void loop(void)
// ----------------------------------------------------------------------------
void loop()
{
  if (RS485.available()) Serial.print(RS485.receive(), HEX);
}
```

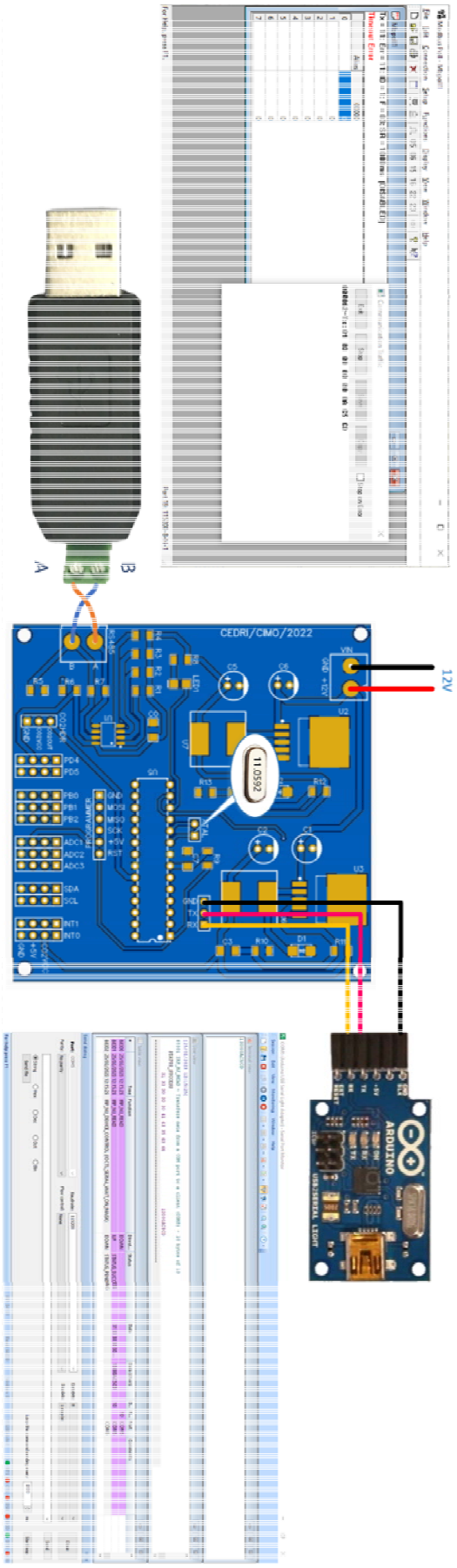


Figura 7 - Ligações realizadas para o teste e depuração da interface de comunicação RS485.

Como se pode ver no esquema da Figura 7, uma bridge USB para RS485 é ligada ao transceiver RS485 da PCB. Por sua vez, no outro extremo, esta bridge liga ao PC que a reconhece como uma porta série. Neste caso em particular, **COM19**.

Utilizando o software Modbus Poll, e configurando-o para comunicar com esta bridge, enviou-se uma sequência de bytes. A Figura 8 mostra a parametrização usada na configuração e a Figura 9 a sua utilização para o envio da sequência:

0x01 0x03 0x00 0x00 0x00 0x0A 0xC5 0xCD

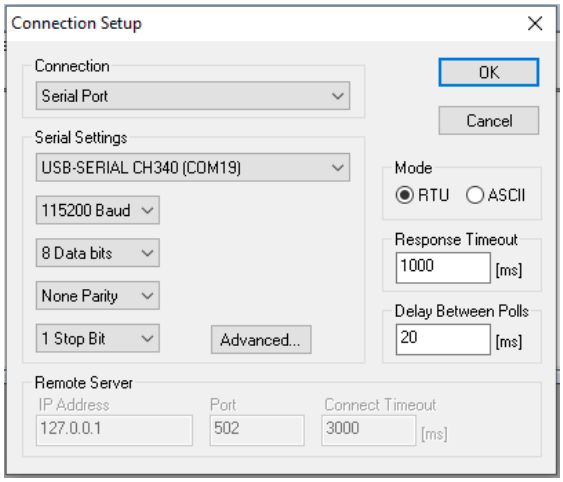


Figura 8 - Configuração do software Modbus Poll.

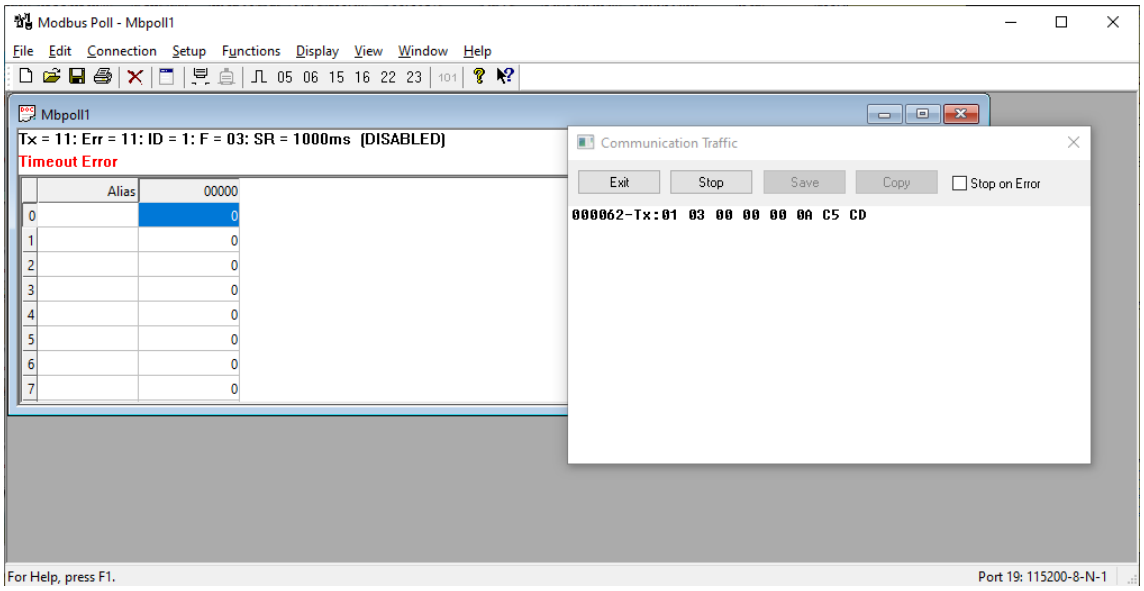


Figura 9 - Screenshot do software Modbus Poll e a sequência de bytes que este enviou.

Foi devido a este ensaio que se percebeu que o baudrate de 115.2 kbps não poderia ser gerado com erro abaixo de um limite aceitável. Sem o cristal de 11.0595MHz, e com o clock interno de 8 MHz, muitos dos carateres enviados pelo Modbus Poll eram recebidos de forma errada.

O sketch apresentado anteriormente tem apenas como função o retorno (eco) dos bytes recebidos agora via RS232. Para isso, uma bridge USB para RS232 é utilizada como também é evidente no diagrama de ligações já apresentado.

No lado do PC, o software Serial Port Monitor é configura de modo a receber os dados da porta série enumerada quando a bridge é ligada (neste caso COM5).

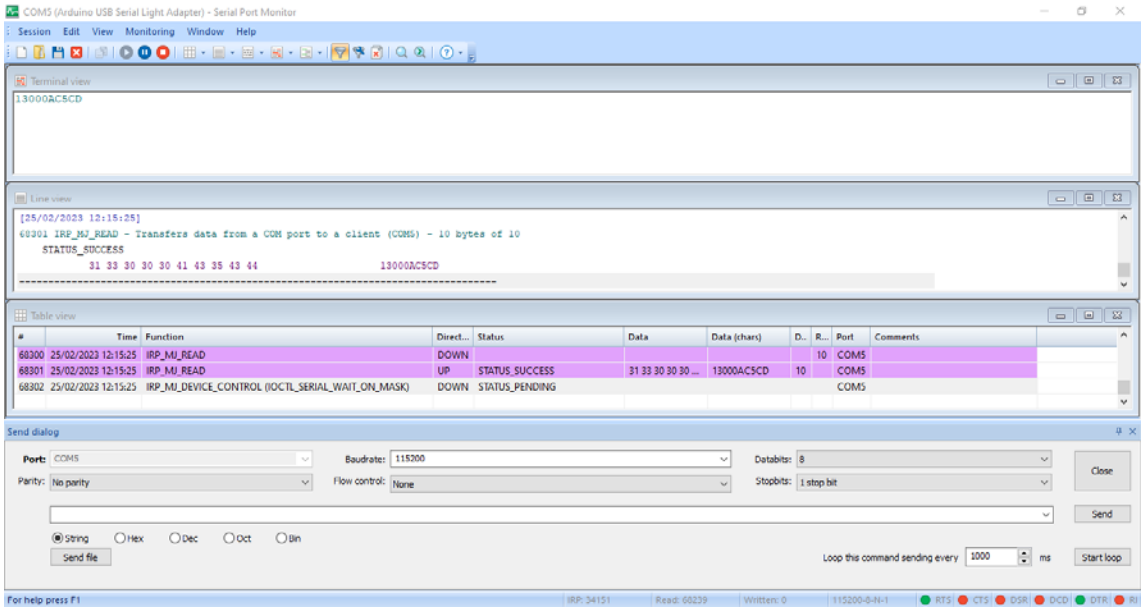


Figura 10 - Resultado observado a partir do Serial Port Monitor.

Como se pode verificar, a sequência enviada e a sequência transmitida são idênticas validando, deste modo, toda a cadeia de hardware e firmware que irá permitir o comando deste módulo de medida.

3.2. Máquina de estados finitos para atendimento ao protocolo

Ainda que deixe o software preparado para facilmente se implementar o protocolo Modbus de forma integral, para este projeto, e por razões de tempo, apenas se considera, para já, a função 03 que se refere à leitura de registos (holding registers).

De forma geral, este código de função envolve um payload (PDU) composto por 5 bytes:

1º byte	2º e 3º byte	4º e 5º byte
Código da função	Endereço base	Numero de registos
03	0x0000 a 0xFFFF	0x0001 a 0x007D

A resposta normal varia em comprimento de 4 bytes até 252 bytes, dependendo do número de registradores de retenção solicitados:

1º byte	2º byte	3º a N bytes
Código da função	Numero de bytes da resposta	Dados enviados
03	2xRegisterCount	…

O número total de bytes retornados é 2 + 2 x RegisterCount, onde RegisterCount é o número de registradores de retenção solicitados. Por exemplo, uma solicitação para 1 registrador de retenção retornará 4 bytes. Uma solicitação de 2 registradores de retenção retornará 6 bytes. Uma solicitação de 125 registradores de retenção retornará 252 bytes.

Após uma solicitação, existem 4 resultados possíveis do escravo.

1. A solicitação é processada com sucesso pelo escravo e uma resposta válida é enviada.
2. O pedido não é recebido pelo escravo, portanto, nenhuma resposta é enviada.
3. A requisição é recebida pelo escravo com erro de paridade, CRC ou LRC. O escravo ignora a requisição e não responde.
4. A requisição é recebida sem erro, mas não pode ser processada pelo escravo por outro motivo. O escravo responde com uma resposta de exceção.

Numa resposta normal, o escravo repete o código de função. O primeiro sinal de uma resposta de exceção é que o código de função é mostrado no eco com seu conjunto de bits mais alto. Todos os códigos de função têm 0 como seu bit mais significativo. Portanto, definir este bit como 1 é o sinal de que o escravo não pode processar a solicitação. A tabela que se segue mostra, à esquerda, os códigos de função possíveis e à direita a exceção que lhe está associada. Em suma, a coluna da direita é obtida da coluna da esquerda invertendo o bit mais dignificativo, ou, em alternativa, somar 0x80.

Função	Exceção
01 (01 hex) 0000 0001	129 (81 hex) 1000 0001
02 (02 hex) 0000 0010	130 (82 hex) 1000 0010
03 (03 hex) 0000 0011	**131 (83 hex) 1000 0011**
04 (04 hex) 0000 0100	132 (84 hex) 1000 0100
05 (05 hex) 0000 0101	133 (85 hex) 1000 0101
06 (06 hex) 0000 0110	**134 (86 hex) 1000 0110**
15 (0F hex) 0000 1111	143 (8F hex) 1000 1111
16 (10 hex) 0001 0000	144 (90 hex) 1001 0000

Código da exceção	Designação	Descrição
01 (01 hex)	Illegal Function	O código de função recebido na consulta não é uma ação permitida para o escravo. Isso pode ocorrer porque o código de função só é aplicável a dispositivos mais novos e não foi implementado na unidade selecionada. Também pode indicar que o escravo está no estado errado para processar uma requisição deste tipo, por exemplo, porque está desconfigurado e está sendo solicitado a retornar valores de registro. Se um comando Poll Program

		Complete foi emitido, esse código indica que nenhuma função do programa o precedeu.
02 (02 hex)	Illegal Data Address	O endereço de dados recebido na consulta não é um endereço permitido para o escravo. Mais especificamente, a combinação de número de referência e comprimento de transferência é inválida. Para um controlador com 100 registradores, uma requisição com offset 96 e comprimento 4 teria sucesso, uma requisição com offset 96 e comprimento 5 irá gerar a exceção 02.
03 (03 hex)	Illegal Data Value	Um valor contido no campo de dados da consulta não é um valor permitido para o escravo. Isso indica uma falha na estrutura do restante de uma solicitação complexa, como o comprimento implícito incorreto. Especificamente, NÃO significa que um item de dado enviado para armazenamento em um registrador tenha um valor fora da expectativa do programa aplicativo, pois o protocolo MODBUS desconhece a significância de qualquer valor específico de qualquer registrador específico.
04 (04 hex)	Slave Device Failure	Ocorreu um erro irrecuperável enquanto o escravo tentava executar a ação solicitada.
05 (05 hex)	Acknowledge	Uso especializado em conjunto com comandos de programação. O escravo aceitou o pedido e está processando-o, mas será necessário um longo período de tempo para fazê-lo. Essa resposta é retornada para evitar que ocorra um erro de tempo limite no mestre. Em seguida, o mestre pode emitir uma mensagem de Poll Program Complete para determinar se o processamento foi concluído.
06 (06 hex)	Slave Device Busy	Uso especializado em conjunto com comandos de programação. O escravo está envolvido no processamento de um comando de programa de longa duração. O mestre deve retransmitir a mensagem mais tarde, quando o escravo estiver livre.
07 (07 hex)	Negative Acknowledge	O escravo não pode executar a função do programa recebida na consulta. Este código é retornado para uma solicitação de programação malsucedida usando o código de função 13 ou 14 decimal. O mestre deve solicitar informações de diagnóstico ou erro do escravo.
08 (08 hex)	Memory Parity Error	Uso especializado em conjunto com os códigos de função 20 e 21 e o tipo de referência 6, para indicar que a área de arquivo estendida não passou em uma verificação de consistência. O escravo tentou ler a memória estendida ou o arquivo de gravação, mas detetou um erro de paridade na memória. O mestre pode repetir a solicitação, mas o serviço pode ser necessário no dispositivo escravo.
10 (0A hex)	Gateway Path Unavailable	O uso especializado em conjunto com gateways indica que o gateway não conseguiu alocar um caminho de comunicação interna da porta de entrada para a porta de saída para processar a solicitação. Geralmente significa que o gateway está configurado incorretamente ou sobrecarregado.
11 (0B hex)	Gateway Target Device Failed to Respond	O uso especializado em conjunto com gateways indica que nenhuma resposta foi obtida do dispositivo de destino. Geralmente significa que o dispositivo não está presente na rede.

Quando um dispositivo escravo Modbus deteta um erro CRC (Cyclic Redundancy Check) em uma mensagem recebida do mestre, normalmente retorna uma resposta de exceção com o código de erro 0x03 (Hex) ou 3 (Decimal), que é conhecido como "Modbus Exception Code 03 - Valor Ilegal de Dados". Este código de erro indica que a mensagem recebida possui um CRC malformado ou incorreto e o escravo não consegue interpretar a mensagem corretamente.

```
#include <SoftwareSerial.h>
#include <TinyRS485.h>
#include <TinyModbus.h>
//
#define MAX485_DE 4
#define MAX485_RE 5
#define MAX485_RO 2
#define MAX485_DI 3
//
#define TIMEOUT 100
#define FRAMELGTH 8
#define MYID 69
#define LIMITADD 0x000E

int bufcnt=0;
const int BUFFER_SIZE = 10;
uint8_t   RX[BUFFER_SIZE]={0,0,0,0,0,0,0,0,0,0}; // Buffer com a mensagem recebida do
master
uint8_t   TX[BUFFER_SIZE]={0,0,0,0,0,0,0,0,0,0}; // Buffer com a mensagem de retorno
para o master
int       TxLen = 0;                              // Dimensão da mensagem a enviar
uint16_t  CRC = 0x0000;                           // CRC da mensagem a enviar
uint16_t  BaseAddress = 0x0000;                   // Endereço base solicitado pelo Master
uint16_t  OfftAddress = 0x0000;                   // Offser associado ao endereço
solicitado pelo Master

unsigned long myTime;

int i = 0; // Índice genérico usado em ciclos FOR

SoftwareSerial SoftSerial(MAX485_RO,MAX485_DI); // RX, TX
TinyRS485 RS485;

enum estadosistema{
  SETRCVMODE, // Coloca o MAX485 no modo de receção
  WAIT4FRAME, // Agurada que uma frame completa seja recebida
  TSTINFRAME, // Verifica se há algum erro com a frame
  MESSAGE4ME, // A mensagem é para mim. Verifica se possuo a função solicitada
  RDFUNCTION,
  WEFUNCTION,
  FUNCNERROR,
  ADDREERROR,
  DECODE4RDE, // Descodifica para envio de dados
  DECODE4WTE, // Descodifica para alteração de variáveis
  ANSWERFRME, // Responde a um pedido
  ERRORFRAME  // Responde com um erro
} estado=SETRCVMODE;

//
bool checkForFrame()
{
  bufcnt=0;
  if (RS485.available())
  {
    myTime = millis();
    RX[bufcnt]=RS485.receive();
    bufcnt++;
    // Baudrate = 115200 bps, 80 bits => 0.7 ms para receber a frame Modbus
    while((millis()-myTime)<TIMEOUT)
    {
      if (RS485.available())
      {
         RX[bufcnt]=RS485.receive();
```

```
        bufcnt++;
      }
    }
  }

  if (bufcnt>=FRAMELGTH) return(true);
  else return(false);
}

//
void setup()
{
  Serial.begin(115200);
  SoftSerial.begin(115200); // Porto série (software) usado pelo transceiver RS485
  pinMode(MAX485_DE,OUTPUT);
  pinMode(MAX485_RE,OUTPUT);
  RS485.begin(SoftSerial,MAX485_DE,MAX485_RE);
}

//
void loop()
{
 //
 switch(estado)
 {
    //
    case SETRCVMODE:
                  while (RS485.available()) RS485.receive(); // Limpa buffer
                  Serial.println("Inicio da maquina de estados");
                  digitalWrite(MAX485_DE,LOW); // Desativa DRIVER
                  digitalWrite(MAX485_RE,LOW); // Ativa RECEIVER
                  estado = WAIT4FRAME;
                  break;
    //
    case WAIT4FRAME:
                  if (checkForFrame()) estado = TSTINFRAME;
                  break;
    //
    case TSTINFRAME:
                  // Mostra frame que recebeu
                  for (i=0;i<bufcnt;i++) Serial.print(RX[i],HEX);
                  Serial.println("");
                  // Verifica integridade...
                  // RX[0]^=0x01; // Simula erro de transmissão (alteração de 1 bit)
                  if(compute_crc(RX,bufcnt)==0){ // CRC OK
                    // Verifica se a frame é para mim...
                    if(RX[0]== MYID){
                      Serial.println("A mensagem E para mim");
                      estado = MESSAGE4ME; // Sim, sou eu...
                    }
                    else{
                      Serial.println("A mensagem NAO e para mim");
                      estado = WAIT4FRAME; // Nop, aguarda por nova frame...
                    }
                  }
                  else{ // ERRO DE TRANSMISSÂO
                    Serial.println("ERRO de CRC");
                    estado = ERRORFRAME;
                  }
                  break;
    //
    case MESSAGE4ME:
                  // Verifica se o slave pode realizar a função solicitada em RX[1]
                  switch(RX[1]) {
                    case 0x03:  estado=RDFUNCTION;
                                Serial.println("Pedem para enviar qualquer coisa");
                                break;
                    case 0x06:  estado=WEFUNCTION;
                                Serial.println("Pedem para alterar qualquer coisa");
                                break;
                    default:    estado=FUNCNERROR;
                                Serial.println("Nao sei o que querem");
                                break;
                  }
                  break;
    //
    case RDFUNCTION:
```

```
                // Verifica se o endereço está dentro dos limites solicitados
                BaseAddress = (((uint16_t)RX[2])<<8)+((uint16_t)RX[3]);
                OfftAddress = (((uint16_t)RX[4])<<8)+((uint16_t)RX[5]);
                if (BaseAddress+OfftAddress>LIMITADD){
                  estado = ADDREERROR;
                  Serial.println("Endereço de leitura fora dos limites");
                }
                else {
                  estado = DECODE4RDE; // Descodifica pedido de leitura
                }
                break;
    //
    case WEFUNCTION:
                // Verifica se o endereço está dentro dos limites solicitados
                BaseAddress = (((uint16_t)RX[2])<<8)+((uint16_t)RX[3]);
                OfftAddress = (((uint16_t)RX[4])<<8)+((uint16_t)RX[5]);
                if (BaseAddress+OfftAddress>LIMITADD){
                  estado = ADDREERROR;
                  Serial.println("Endereco de escrita fora dos limites");
                }
                else {
                  estado = DECODE4WTE; // Descodifica pedido de escrita
                }
                break;
    //
    case FUNCNERROR:
                // Constroi frame
                Serial.println("Retorna info de Function Error");
                TX[0] = MYID;
                TX[1] = RX[1] + 0x80;
                TX[2] = 0x01;
                TxLen = 3;
                estado = ANSWERFRME;
                break;
    //
    case ADDREERROR:
                // Constroi frame
                Serial.println("Retorna info de Address Error");
                TX[0] = MYID;
                TX[1] = RX[1] + 0x80;
                TX[2] = 0x02;
                TxLen = 3;
                estado = ANSWERFRME;
                break;
    //
    case DECODE4RDE:
                Serial.println("Descodifico frame e executo pedido");
                // Coleta a informação necessária para originar a resposta
                // Constroi frame
                TX[0] = MYID;
                TX[1] = RX[1];
                TX[2] = 0x02;
                TX[3] = 0xAB;
                TX[4] = 0xCD;
                TxLen = 5;
                // Rotina para execução do pedido.
                estado = ANSWERFRME;
                break;
    //
    case ANSWERFRME:
                 Serial.println("Resposta a Frame (via RS485)");
                // Adiciona CRC à frame
                CRC   = compute_crc(TX,TxLen);
                TX[TxLen++] = (uint8_t) CRC;
                TX[TxLen++] = (uint8_t) (CRC>>8);
                // Ativa MAX485 para TX
                digitalWrite(MAX485_DE,HIGH); // Ativa DRIVER
                digitalWrite(MAX485_RE,HIGH); // Desativa RECEIVER
                // Transmite dados...
                for(i=0;i<TxLen;i++) RS485.transmit(TX[i]);
                estado=SETRCVMODE;
                break;
    //
    case ERRORFRAME:
                Serial.println("ERRO: CRC inválido");
                TX[0] = MYID;
                TX[1] = RX[1] + 0x80;
```

```
                    TX[2] = 0x03;
                    TxLen = 3;
                    estado = ANSWERFRME;
                    break;
    //
  }
}
```

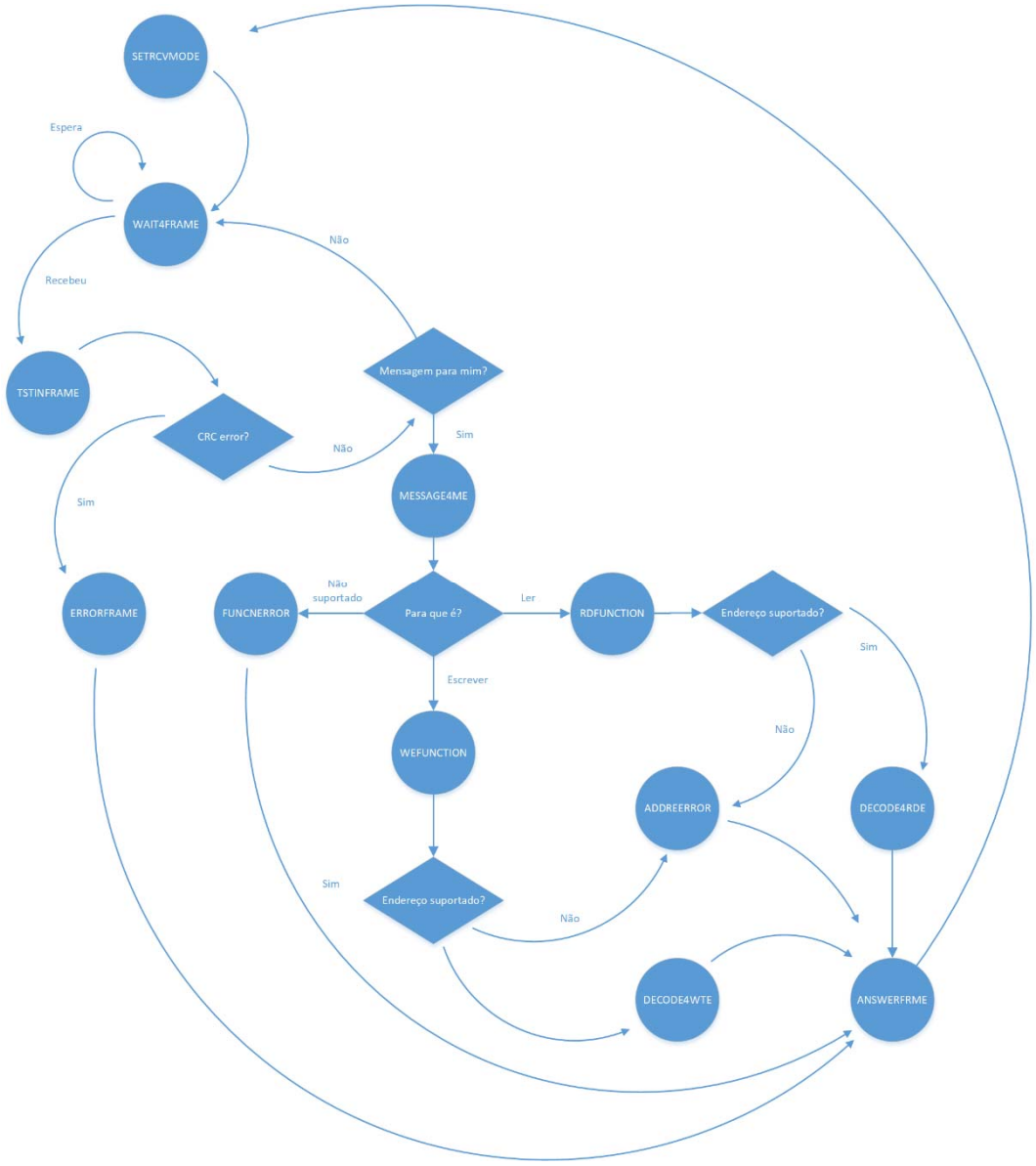


Figura 11 - Máquina de estados executada pelo sketch.

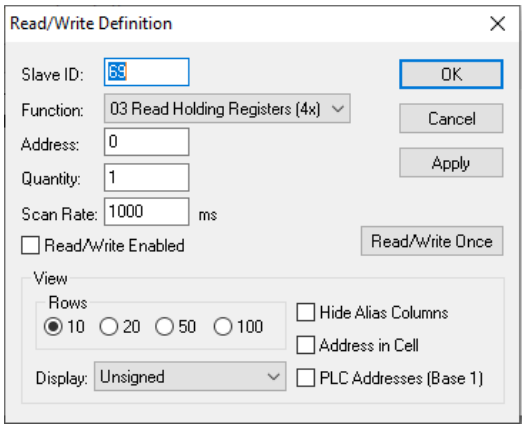


Figura 12 - Parametrização do Modbus Poll. Note-se o preenchimento do endereço do Slave (base 10).

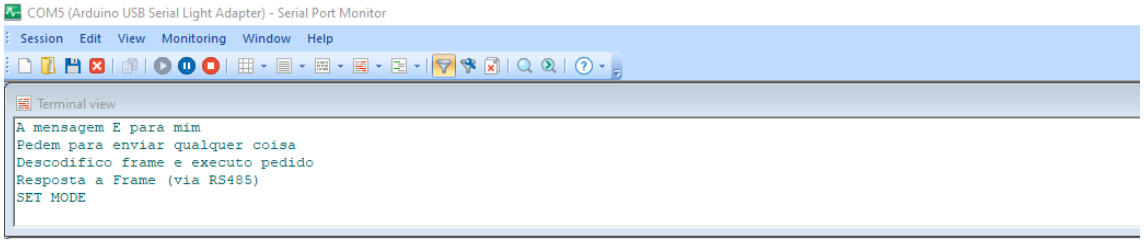


Figura 13 - Quadro das mensagens de debug enviadas pelo sistema.

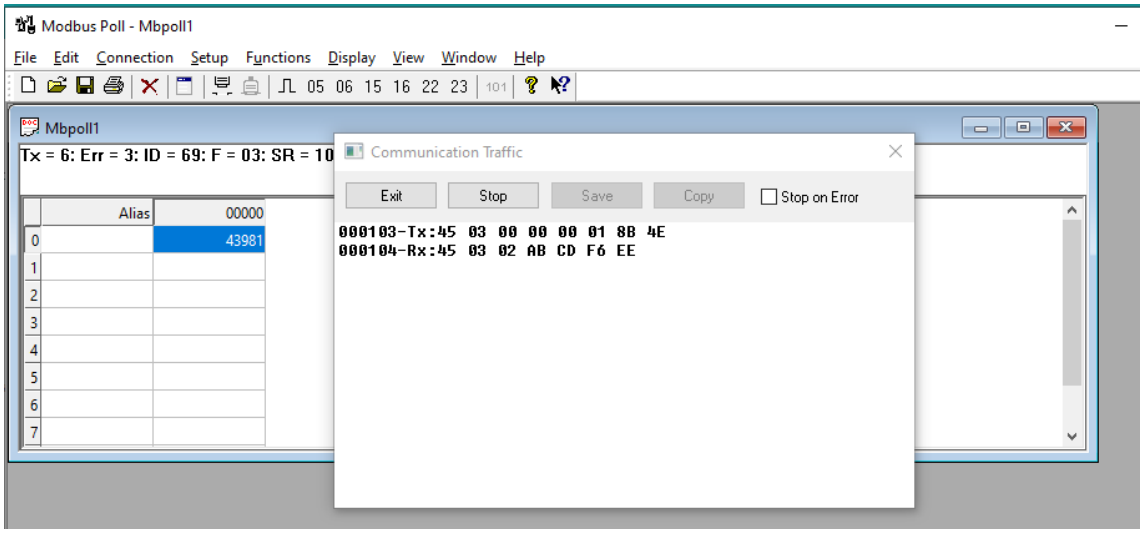


Figura 14 - Resultado do envio e resposta do módulo de CO2.

Exemplo de erro

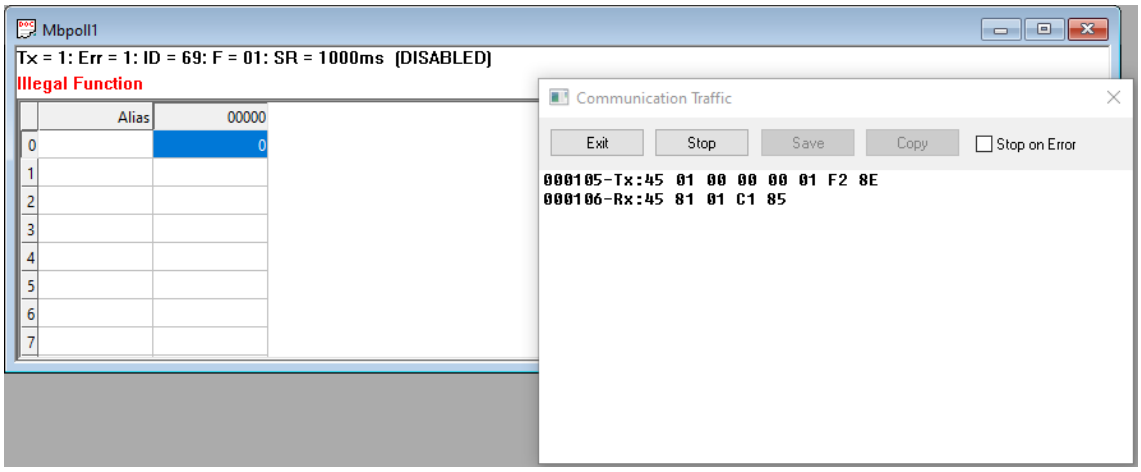


Figura 15 - Resultado de solicitação de uma função de código não considerada no firmware.

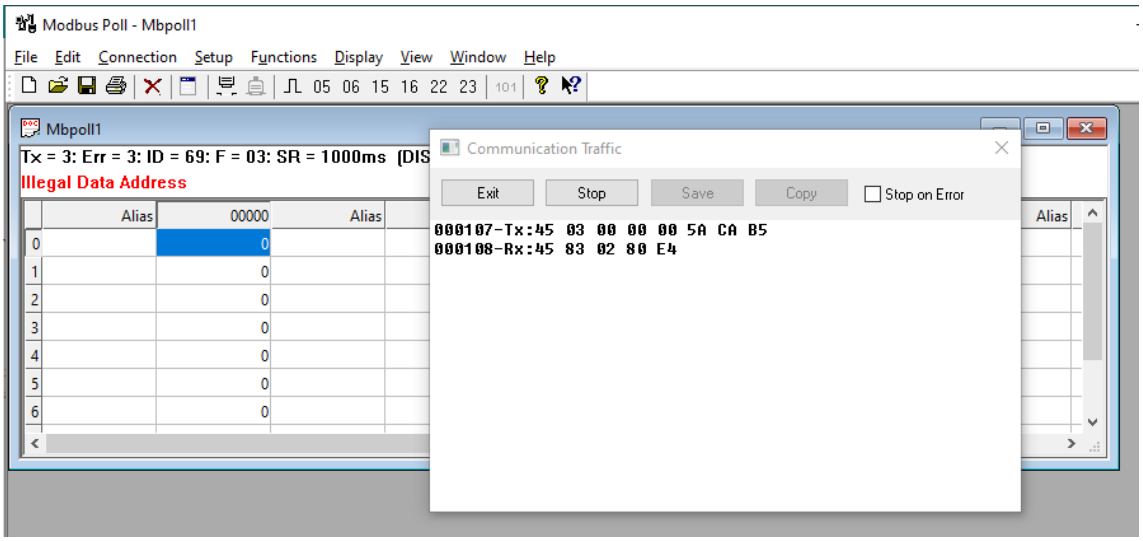


Figura 16 - Resultado da solicitação de um intervalo de endereços inválido.

4. Parametrização das mensagens

Parâmetro	Endereço do Registo	Função	Gama	Valor por defeito (REG)	Valor por defeito (FIRM)
DC_GAIN	0x0000	3,6	GAIN = (float) Valor do registo/100.0	0x0352	8.5
READ_SAMPLE_INTERVAL	0x0001	3,6	SINT = (unsigned int) Valor do registo	0x0032	50
READ_SAMPLE_TIMES	0x0002	3,6	STIM = (unsigned int) Valor do registo	0x0005	5
ZERO_POINT_VOLTAGE	0x0003	3,6	ZPV = (float) Valor do registo /10 000.0	0x0898	0.220
REACTION_VOLTAGE	0x0004	3,6	RVT = (float)) Valor do registo /10 000.0	0x012C	0.030
Modbus Slave Address	0x0007	3, 6		0x0045	0x0045
CO2_PPM	**0x0008**	**3**	**CO2 = Valor do registo**	**N/A**	

TEMPERATURA_AR	0x0009	3	TAR = Valor do registo		
HUMIDADE_AR	0x000A	3	HAR = Valor do registo		

Neste momento, apenas os três últimos registos serão implementados.

Exemplo: Ler o valor do CO2

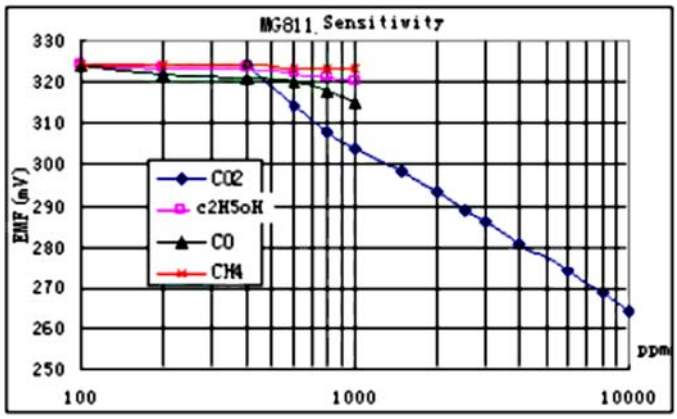
Solicitação: 45 03 00 08 00 01 **0A 8C**

Resposta: 45 03 02 **01 CA C9 8C**

Neste caso, a resposta indica que o valor da concentração de CO_2 é de 458 ppm.

5. Sensor de CO2 (MG811)

Symbol	Parameter Name	Technical	Remarks
V_H	Heating Voltage	6.0±0.1 V	AC or DC
R_H	Heating Resistor	30.0±5%Ω	Room Temperature
I_H	Heating Current	@200mA	
P_H	Heating Power	@1200mW	
Tao	Operating Temperature	-20—50	
Tas	Storage Temperature	-20—70	
? E?M F	Output	30—50mV	350—10000ppmCO2



Curva de calibração considerando uma temperatura de referência de 28ºC, humidade relativa de 65% e concentração de oxigénio igual a 21%.

Piecewise linear no domínio logarítmico

	Equação 1		Equação 2	
EMF/mV	324	307	307	265
CO_2/ppm	400	800	800	10000

Todas as equações do tipo:

$$y = \lambda \cdot 10^{\gamma x} \tag{0.1}$$

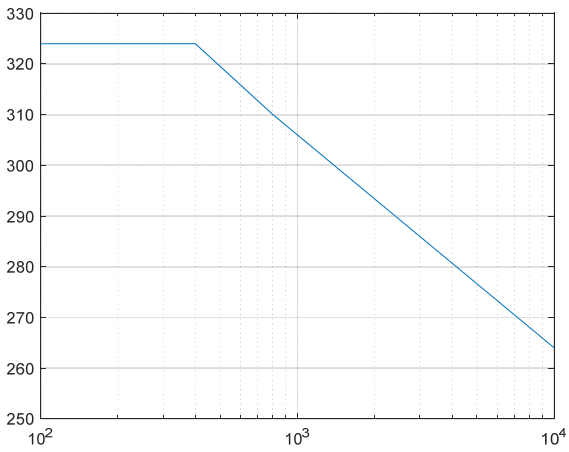
Levam a retas num gráfico semilogarítmico. Aplicando logaritmos a ambos os termos resulta em:

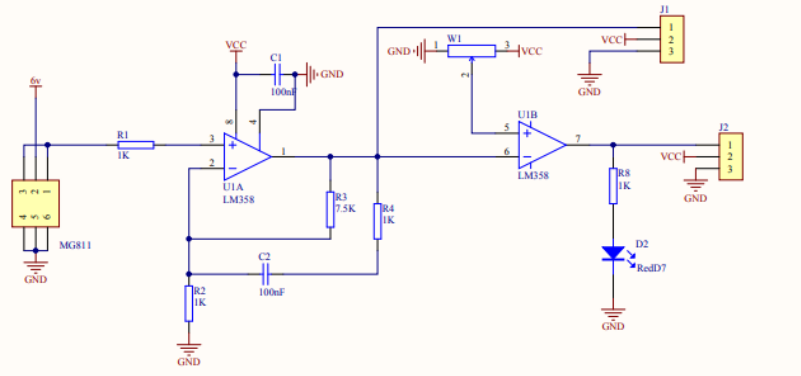
$$\log_{10}(y) = \gamma x + \log_{10}(\lambda) \tag{0.2}$$

$$\begin{cases} \gamma = \dfrac{324-307}{400-800} = -0.0425 \\ \log_{10}(\lambda) = 19.5105 \end{cases} \tag{0.3}$$

Neste caso,

```
% Função de calibração do sensor MG811
clear all;
close all;
clc;

y1=[324 307];
x1=[400 800];

m1=-46.3
b1=444.4841

y2=[307 265];
x2=[1000 10000];

m2=y2(2)-y2(1)
b2=432;

N=50000;
CO2=linspace(100,10000,N);
k=0;
for i=1:N
    if (CO2(i)<400)
        EMF(i)=324;
    elseif (CO2(i)>=400 && CO2(i)<800)
        if(k==0) disp(num2str(b1 + m1*log10(CO2(i)))); end
        k=1;
        EMF(i) = b1 + m1*log10(CO2(i));
    elseif (CO2(i)>=800)
        EMF(i) = b2 + m2*log10(CO2(i));
    end
end

semilogx(CO2,EMF);
axis([100 10000 250 330])
grid on
```





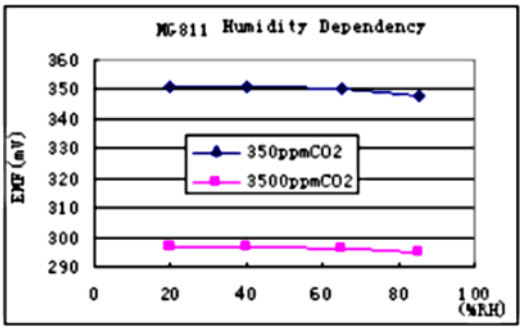
Amplificador com ganho igual a 7.5. Quando Vo = 324mV, Vout=2.43V

- AT = 50ºC
- AV=10mV
- 0.2mV/ºC

Compensação da temperatura….

- T=28 => K=1
- T=-10 => K=343/338 = 1.0148
- T=50 => K=343/345=0.9942

K = -3.8833e-04 * T + 1.0109



5.1. Máquina de estado para controlo do sensor de CO_2

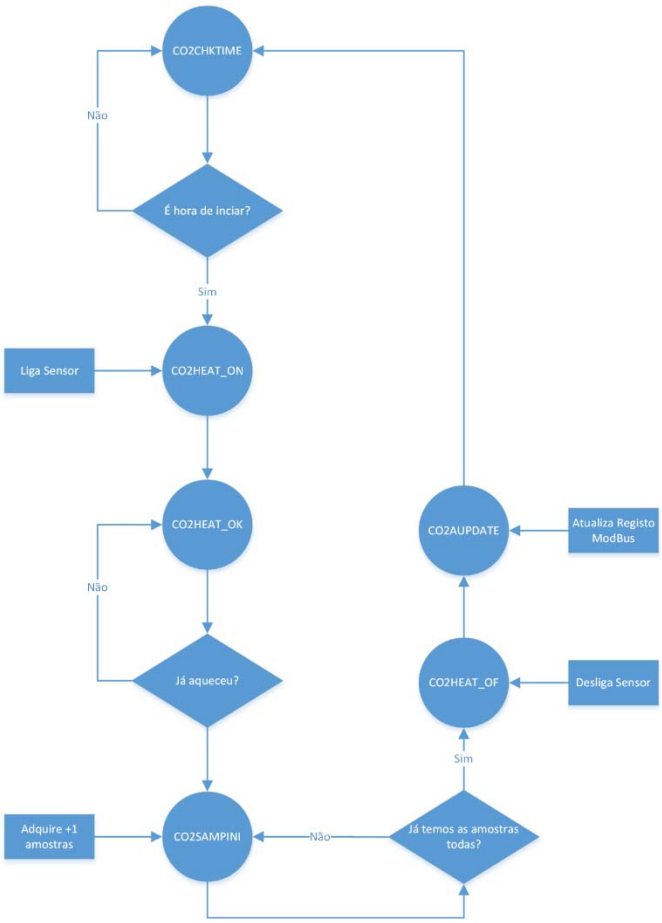
A cada hora é realizado a medição do nível de CO2. Para isso, o sensor é ativado durante 5 minutos para se realizar o aquecimento. Ao fim desses 5 minutos, é feita a aquisição do valor do CO_2, em ppm, durante 1 minuto em intervalos de 1 segundo. O valor armazenado no registo, e disponibilizada via solicitação por Modbus, é a média dessas sessenta amostras.

O sketch que se apresenta a seguir descreve a máquina de estados que deve ser executada para se proceder à medição do nível de CO2.

```
#include <SoftwareSerial.h>
#include <TinyRS485.h>
#include <TinyModbus.h>
#include "SEN0159.h"
#include <EEPROM.h> // Utilizado no futuro para guardar novo ID do slave e parâmetros
dos sensores...


// Termos associados à medição do CO2
// MG_PIN - Sensor de CO2 ligado à porta A/D A0
//#define CO2Vcc                PD7        // Fonte de alimentação para o sensor de CO2
(Ativa baixa).
//#define CO2MeasurementUpdate  1*3600000  // Período de amostragem para a medição do
CO2 (em horas convertidas para millisegundos)
//#define CO2HeatingTime        5*60000    // Tempo necessário para pré-aquecer o sensor
//#define CO2TimeBetweenSamples 1*1000     // Tempo utilizado entre amostras
consecutivas do CO2 (filtragem)
//#define CO2FilterLenght       1*60       // Numero de amostras necessárias para filtro
de média
//#define LED                   PD6        // LED ligado ao pino 12 do ATMEGA328 (PD6)
(ativa-baixa)

// Para testes    |------|---|---|------|------|------|
// tempo (min)    0      1  1.5  2      3      4      5
// Duty cycle 1.5/5 = 30%

#define CO2Vcc                PD7        // Fonte de alimentação para o sensor de CO2
(Ativa baixa).
#define CO2MeasurementUpdate  300000     // Período de amostragem para a medição do CO2
(5 minutos)
#define CO2HeatingTime        60000      // Tempo necessário para pré-aquecer o sensor
(1 minuto)
#define CO2TimeBetweenSamples 500        // Tempo utilizado entre amostras consecutivas
(1/2 segundo)
#define CO2FilterLenght       1*60       // Numero de amostras necessárias para filtro
de média
#define LED                   PD6        // LED ligado ao pino 12 do ATMEGA328 (PD6)
(ativa-baixa)


//
uint16_t REGS[3] = {0, 0, 0}; // Registo associado às observações
//
unsigned long myTime;
unsigned long LastCO2Measurement = 0; // Retem o tempo em que foi realizada a última
medição do CO2
unsigned long LastHeaterTime = 0;     // Retem o tempo para aquecimento do sensor
unsigned long LastCO2Sampling = 0;    // Retem o tempo em que foi realizada a última
amostragem do CO2
int CO2SampleCounter = 0;             // Contador para manter registo das amostras
adquiridas.
float CO2filtrado = 0.0;                   // Retém valor relativo à soma das amostras
do CO2
uint16_t CO2SUM[CO2FilterLenght];
int i = 0; // Índice genérico usado em ciclos FOR


// ------------------------------------------------------------------------------------
----------------------------
// MÁQUINA DE ESTADOS PRINCIPAL
// ------------------------------------------------------------------------------------
----------------------------
enum co2sensorstate {
  CO2CHKTIME, // Verifica se é altura para iniciar novo sequencia de amostragem
  CO2HEAT_ON, // Liga sensor para iniciar processo de aquecimento
  CO2HEAT_OF, // Desliga sensor
  CO2HEAT_OK, // Tempo de aquecimento completo
  CO2SAMPLER, // Adquire nova amostra
  CO2SAMPINI, // Inicia processo de amostragem
  CO2AUPDATE, // Atualiza registos e variáveis de estado
  CO2SPERROR  // Erro na amostragem
} co2estado = CO2HEAT_ON;


// ------------------------------------------------------------------------------------
----------------------------
// void setup(void)
```

```
// ------------------------------------------------------------------------------------------
----------------------------
void setup()
{
  Serial.begin(115200);
  pinMode(LED, OUTPUT);        // LED para debug ou sinalização
  pinMode(CO2Vcc, OUTPUT);     // Estabelece porto para controlo de CO2Vcc
  digitalWrite(CO2Vcc, HIGH);  // Desliga fonte de alimentação do sensor de CO2
  digitalWrite(LED, HIGH);     // Desliga fonte de alimentação do sensor de CO2
  LastCO2Sampling = millis();  // Atualiza timetag
}

// ------------------------------------------------------------------------------------------
----------------------------
// void loop(void)
// ------------------------------------------------------------------------------------------
----------------------------
void loop()
{
  //
..................................................................................
...........
  // Máquina de estados
  //
..................................................................................
...........
  switch (co2estado)
  {
    //
..................................................................................
.........
    case CO2CHKTIME:
      if ((millis() - LastCO2Measurement) >= CO2MeasurementUpdate) {
        LastCO2Measurement = millis();
        co2estado = CO2HEAT_ON;
      }
      break;
    //
..................................................................................
.........
    case CO2HEAT_ON:
      Serial.println("Inicia aquecimento do sensor");
      digitalWrite(CO2Vcc, LOW);   // Liga fonte de alimentação do sensor de CO2
      digitalWrite(LED, LOW);      // Liga LED
      LastHeaterTime = millis();
      co2estado = CO2HEAT_OK;
      break;
    //
..................................................................................
.........
    case CO2HEAT_OK:
      if ((millis() - LastHeaterTime) >= CO2HeatingTime) {
        Serial.println("Aquecimento finalizado... amostrando");
        //CO2SUM = 0.0;
        LastCO2Sampling = millis();
        co2estado = CO2SAMPINI;
      }
      break;
    //
..................................................................................
.........
    case CO2SAMPINI:
      if (CO2SampleCounter < CO2FilterLenght) { // Ainda não foram obtidas todas as
amostras necessárias...
        if ((millis() - LastCO2Sampling) >= CO2TimeBetweenSamples) {
          CO2SUM[CO2SampleCounter]= (uint16_t) CO2ppm(MG_PIN, CO2Curve);
          CO2SampleCounter++;
          LastCO2Sampling = millis();
        }
      }
      else { // Já estão todas as amostras adquiridas...
        Serial.print("Numero de amostras = ");
        Serial.println(CO2SampleCounter);
        CO2SampleCounter = 0;   // Reinicia contador de amostras
        co2estado = CO2HEAT_OF; // Manda desligar o sensor
      }
      break;
```

```
    //
.....................................................................................
.........
    case CO2HEAT_OF:
      Serial.println("Desliga sensor");
      digitalWrite(CO2Vcc, HIGH);   // Desliga fonte de alimentação do sensor de CO2
      digitalWrite(LED, HIGH);      // Desliga LED
      co2estado = CO2AUPDATE;
      break;
    //
.....................................................................................
.........
    case CO2AUPDATE:
      //REGS[0] = (uint16_t)(CO2SUM / (float)CO2FilterLenght); // Coloca valor no
registo
      //Serial.print("CO2 (ppm) = ");
      //Serial.println(REGS[0]);
       CO2filtrado=0.0;
      for (int i=0;i<CO2FilterLenght;i++){
        Serial.println(CO2SUM[i]);
        CO2filtrado+=float(CO2SUM[i]);
      }
      Serial.print ("Valor Filtrado = ");
      Serial.println(CO2filtrado/((float)CO2FilterLenght));
      co2estado = CO2CHKTIME;
      break;
    //
.....................................................................................
.........
    case CO2SPERROR:

      break;
  }
}
```

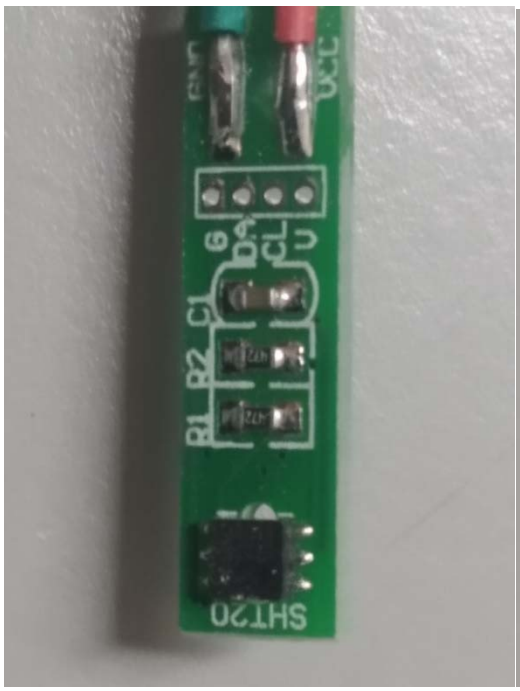


6. Sensor de Humidade e Temperatura (SHT20)

Detalhes adicionais sobre este sensor podem ser encontrado no Wiki.

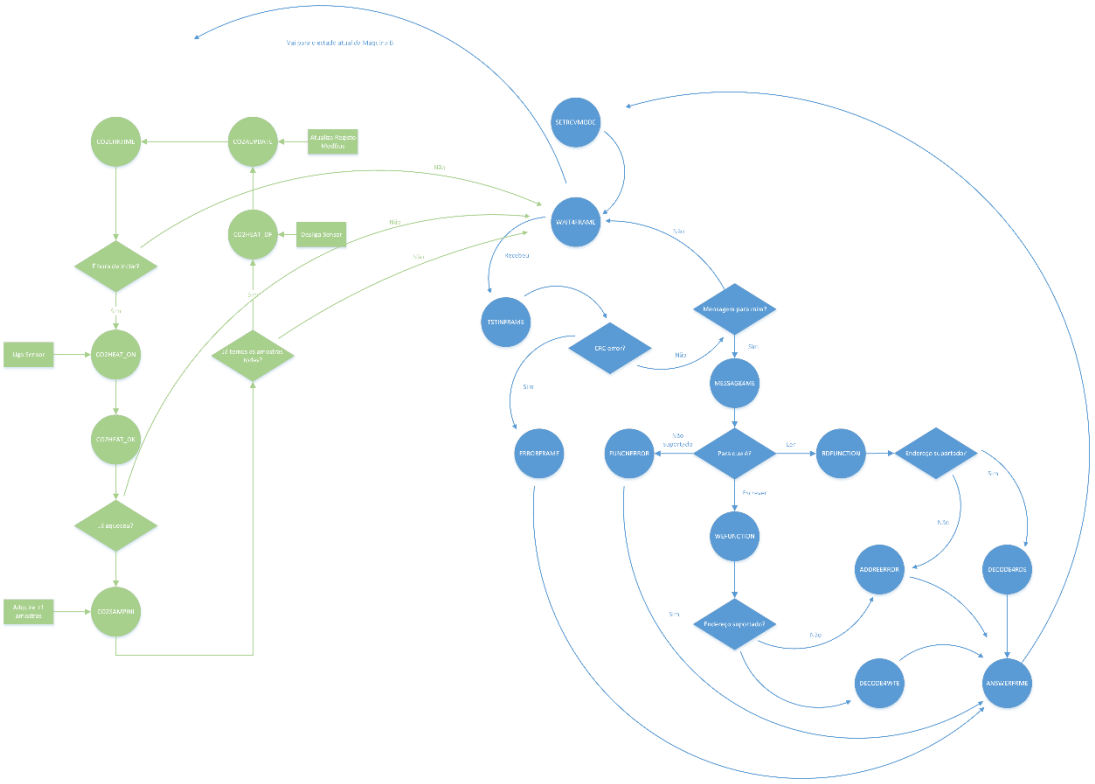
Número	Cor do condutor	Descrição
1	Vermelho	VCC
2	Verde	GND
3	Azul	SDA
4	Amarelo	SCL

Fotos tiradas do sensor adquirido



7. Firmware

7.1. União das duas máquinas de estados:



```
#include <SoftwareSerial.h>
#include <TinyRS485.h>
#include <TinyModbus.h>
#include "SEN0159.h"
#include <EEPROM.h> // Utilizado no futuro para guardar novo ID do slave e parâmetros
dos sensores...

// Termos associados à medição do CO2
// MG_PIN - Sensor de CO2 ligado à porta A/D A0
//#define CO2Vcc                PD7        // Fonte de alimentação para o sensor de CO2
(Ativa baixa).
//#define CO2MeasurementUpdate  1*3600000  // Período de amostragem para a medição do
CO2 (em horas convertidas para millisegundos)
//#define CO2HeatingTime        5*60000    // Tempo necessário para pré-aquecer o sensor
//#define CO2TimeBetweenSamples 1*1000     // Tempo utilizado entre amostras
consecutivas do CO2 (filtragem)
//#define CO2FilterLenght       1*60       // Numero de amostras necessárias para filtro
de média
//#define LED                   PD6        // LED ligado ao pino 12 do ATMEGA328 (PD6)
(ativa-baixa)

// Para testes    |------|---|---|------|------|------|
// tempo (min)    0      1  1.5  2      3      4      5
// Duty cycle 1.5/5 = 30%

#define CO2Vcc                PD7        // Fonte de alimentação para o sensor de CO2
(Ativa baixa).
#define CO2MeasurementUpdate  300000     // Período de amostragem para a medição do CO2
(5 minutos)
#define CO2HeatingTime        60000      // Tempo necessário para pré-aquecer o sensor
(1 minuto)
#define CO2TimeBetweenSamples 500        // Tempo utilizado entre amostras consecutivas
(1/2 segundo)
#define CO2FilterLenght       1*60       // Numero de amostras necessárias para filtro
de média
#define LED                   PD6        // LED ligado ao pino 12 do ATMEGA328 (PD6)
(ativa-baixa)

// Termos associados à comunicação ModBus
#define MAX485_DE 4
#define MAX485_RE 5
#define MAX485_RO 2
#define MAX485_DI 3
//
#define TIMEOUT 100
#define FRAMELGTH 8
#define MYID 69
#define LIMITADD 0x000E

// Variáveis associadas ao CO2
//
uint16_t REGS[3] = {0, 0, 0}; // Registo associado às observações
//
unsigned long myTimeCO2, MyTimeMB;
unsigned long LastCO2Measurement = 0; // Retem o tempo em que foi realizada a última
medição do CO2
unsigned long LastHeaterTime = 0;     // Retem o tempo para aquecimento do sensor
unsigned long LastCO2Sampling = 0;    // Retem o tempo em que foi realizada a última
amostragem do CO2
int CO2SampleCounter = 0;             // Contador para manter registo das amostras
adquiridas.
float CO2filtrado = 0.0;                   // Retém valor relativo à soma das amostras
do CO2
uint16_t CO2SUM[CO2FilterLenght];

// Variáveis associadas ao ModBus
int bufcnt = 0;
const int BUFFER_SIZE = 10;
uint8_t   RX[BUFFER_SIZE] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; // Buffer com a mensagem
recebida do master
uint8_t   TX[BUFFER_SIZE] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; // Buffer com a mensagem de
retorno para o master
int       TxLen = 0;                           // Dimensão da mensagem a enviar
uint16_t  CRC = 0x0000;                        // CRC da mensagem a enviar
```

```
uint16_t  BaseAddress = 0x0000;                  // Endereço base solicitado pelo Master
uint16_t  OfftAddress = 0x0000;                  // Offser associado ao endereço
solicitado pelo Master

int i = 0; // Índice genérico usado em ciclos FOR

SoftwareSerial SoftSerial(MAX485_RO, MAX485_DI); // RX, TX
TinyRS485 RS485;

// ------------------------------------------------------------------------------------------
----------------------------
// MÁQUINA DE ESTADOS
// ------------------------------------------------------------------------------------------
----------------------------
enum estadosistema {
  SETRCVMODE, // Coloca o MAX485 no modo de receção
  WAIT4FRAME, // Agurada que uma frame completa seja recebida
  TSTINFRAME, // Verifica se há algum erro com a frame
  MESSAGE4ME, // A mensagem é para mim. Verifica se possuo a função solicitada
  RDFUNCTION,
  WEFUNCTION,
  FUNCNERROR,
  ADDREERROR,
  DECODE4RDE, // Descodifica para envio de dados
  DECODE4WTE, // Descodifica para alteração de variáveis
  ANSWERFRME, // Responde a um pedido
  ERRORFRAME, // Responde com um erro
  CO2CHKTIME, // Verifica se é altura para iniciar novo sequencia de amostragem
  CO2HEAT_ON, // Liga sensor para iniciar processo de aquecimento
  CO2HEAT_OF, // Desliga sensor
  CO2HEAT_OK, // Tempo de aquecimento completo
  CO2SAMPINI, // Inicia processo de amostragem
  CO2AUPDATE, // Atualiza registos e variáveis de estado
  CO2SPERROR  // Erro na amostragem
};
estadosistema co2estado = CO2CHKTIME;
estadosistema estado = SETRCVMODE;

// ------------------------------------------------------------------------------------------
----------------------------
// bool checkForFrame(void)
// ------------------------------------------------------------------------------------------
----------------------------
bool checkForFrame()
{
  bufcnt = 0;
  if (RS485.available())
  {
    MyTimeMB = millis();
    RX[bufcnt] = RS485.receive();
    bufcnt++;
    // Baudrate = 115200 bps, 80 bits => 0.7 ms para receber a frame Modbus
    while ((millis() -  MyTimeMB) < TIMEOUT)
    {
      if (RS485.available())
      {
        RX[bufcnt] = RS485.receive();
        bufcnt++;
      }
    }
  }
  // O tempo terminou desde que foi recebido o primeiro caracter... verificar se a frame
completa foi recebida
  if (bufcnt >= FRAMELGTH) return (true);
  else return (false);
}

// ------------------------------------------------------------------------------------------
----------------------------
// void setup(void)
// ------------------------------------------------------------------------------------------
----------------------------
void setup()
{
  Serial.begin(115200);
  // ModBus
  SoftSerial.begin(115200); // Porto série (software) usado pelo transceiver RS485
```

```
  pinMode(MAX485_DE, OUTPUT);
  pinMode(MAX485_RE, OUTPUT);
  RS485.begin(SoftSerial, MAX485_DE, MAX485_RE);
  // CO2
  pinMode(LED, OUTPUT);        // LED para debug ou sinalização
  pinMode(CO2Vcc, OUTPUT);     // Estabelece porto para controlo de CO2Vcc
  digitalWrite(CO2Vcc, HIGH);  // Desliga fonte de alimentação do sensor de CO2
  digitalWrite(LED, HIGH);     // Desliga fonte de alimentação do sensor de CO2
  LastCO2Sampling = millis();  // Atualiza timetag
}

// ------------------------------------------------------------------------------------
----------------------------
// void loop(void)
// ------------------------------------------------------------------------------------
----------------------------
void loop()
{

  switch (estado)
  {
    //
.....................................................................................
...........
    // Máquina de estados ModBus
    //
.....................................................................................
...........
    case SETRCVMODE:
      while (RS485.available()) RS485.receive(); // Limpa buffer
      Serial.println("Inicio da maquina de estados");
      digitalWrite(MAX485_DE, LOW); // Desativa DRIVER
      digitalWrite(MAX485_RE, LOW); // Ativa RECEIVER
      estado = WAIT4FRAME;
      break;
    //
.....................................................................................
.........
    case WAIT4FRAME:
      if (checkForFrame()) estado = TSTINFRAME;
      else estado = co2estado;
      break;
    //
.....................................................................................
.........
    case TSTINFRAME:
      // Mostra frame que recebeu
      for (i = 0; i < bufcnt; i++) Serial.print(RX[i], HEX);

      Serial.println("");
      Serial.print("MyID: ");

      Serial.println(RX[0], DEC);
      // Verifica integridade...
      // RX[0]^=0x01; // Simula erro de transmissão (alteração de 1 bit)
      if (compute_crc(RX, bufcnt) == 0) { // CRC OK
        // Verifica se a frame é para mim...
        if (RX[0] == MYID) {
          Serial.println("A mensagem E para mim");
          estado = MESSAGE4ME; // Sim, sou eu...
        }
        else {
          Serial.println("A mensagem NAO e para mim");
          estado = WAIT4FRAME; // Nop, aguarda por nova frame...
        }
      }
      else { // ERRO DE TRANSMISSÂO
        Serial.println("ERRO de CRC");
        estado = ERRORFRAME;
      }
      break;
    //
.....................................................................................
.........
    case MESSAGE4ME:
      // Verifica se o slave pode realizar a função solicitada em RX[1]
      switch (RX[1]) {
```

```
        case 0x03:  estado = RDFUNCTION;
          Serial.println("Pedem para enviar qualquer coisa");
          break;
        case 0x06:  estado = WEFUNCTION;
          Serial.println("Pedem para alterar qualquer coisa");
          break;
        default:    estado = FUNCNERROR;
          Serial.println("Nao sei o que querem");
          break;
      }
      break;
    //
.....................................................................................
.........
    case RDFUNCTION:
      // Verifica se o endereço está dentro dos limites solicitados
      BaseAddress = (((uint16_t)RX[2]) << 8) + ((uint16_t)RX[3]); // Calcula endereço
base
      OfftAddress = (((uint16_t)RX[4]) << 8) + ((uint16_t)RX[5]); // Calcula offset
      if (BaseAddress + OfftAddress > LIMITADD) {
        estado = ADDREERROR;
        Serial.println("Endereço de leitura fora dos limites");
      }
      else {
        estado = DECODE4RDE; // Descodifica pedido de leitura
      }
      break;
    //
.....................................................................................
.........
    case WEFUNCTION:
      // Verifica se o endereço está dentro dos limites solicitados
      BaseAddress = (((uint16_t)RX[2]) << 8) + ((uint16_t)RX[3]); // Calcula endereço
base
      OfftAddress = (((uint16_t)RX[4]) << 8) + ((uint16_t)RX[5]); // Calcula offset
      if (BaseAddress + OfftAddress > LIMITADD) {
        estado = ADDREERROR;
        Serial.println("Endereco de escrita fora dos limites");
      }
      else {
        estado = DECODE4WTE; // Descodifica pedido de escrita
      }
      break;
    //
.....................................................................................
.........
    case FUNCNERROR:
      // Constroi frame
      Serial.println("Retorna info de Function Error");
      TX[0] = MYID;
      TX[1] = RX[1] + 0x80;
      TX[2] = 0x01;
      TxLen = 3;
      estado = ANSWERFRME;
      break;
    //
.....................................................................................
.........
    case ADDREERROR:
      // Constroi frame
      Serial.println("Retorna info de Address Error");
      TX[0] = MYID;
      TX[1] = RX[1] + 0x80;
      TX[2] = 0x02;
      TxLen = 3;
      estado = ANSWERFRME;
      break;
    //
.....................................................................................
.........
    case DECODE4RDE:
      Serial.println("Descodifico frame e executo pedido");
      // Coleta a informação necessária para originar a resposta
      // Constroi frame
      TX[0] = MYID;
      TX[1] = RX[1];
      TX[2] = 0x02;
```

```
      TX[3] = 0xAB;
      TX[4] = 0xCD;
      TxLen = 5;
      // Rotina para execução do pedido.
      estado = ANSWERFRME;
      break;
    //
.....................................................................................
.........
    case ANSWERFRME:
      Serial.println("Resposta a Frame (via RS485)");
      // Adiciona CRC à frame
      CRC   = compute_crc(TX, TxLen);
      TX[TxLen++] = (uint8_t) CRC;
      TX[TxLen++] = (uint8_t) (CRC >> 8);
      // Ativa MAX485 para TX
      digitalWrite(MAX485_DE, HIGH); // Ativa DRIVER
      digitalWrite(MAX485_RE, HIGH); // Desativa RECEIVER
      // Transmite dados...
      for (i = 0; i < TxLen; i++) RS485.transmit(TX[i]);
      estado = SETRCVMODE;
      break;
    //
.....................................................................................
.........
    case ERRORFRAME:
      Serial.println("ERRO: CRC inválido");
      TX[0] = MYID;
      TX[1] = RX[1] + 0x80;
      TX[2] = 0x03;
      TxLen = 3;
      estado = ANSWERFRME;
      break;
    //
.....................................................................................
...........
    // Máquina de estados CO2
    //
.....................................................................................
...........
    case CO2CHKTIME:
      if ((millis() - LastCO2Measurement) >= CO2MeasurementUpdate) {
        LastCO2Measurement = millis();
        co2estado = CO2HEAT_ON;
      }
      estado = WAIT4FRAME;
      break;
    //
.....................................................................................
.........
    case CO2HEAT_ON:
      Serial.println("Inicia aquecimento do sensor");
      digitalWrite(CO2Vcc, LOW);   // Liga fonte de alimentação do sensor de CO2
      digitalWrite(LED, LOW);      // Liga LED
      LastHeaterTime = millis();
      co2estado = CO2HEAT_OK;
      estado = WAIT4FRAME;
      break;
    //
.....................................................................................
.........
    case CO2HEAT_OK:
      if ((millis() - LastHeaterTime) >= CO2HeatingTime) {
        Serial.println("Aquecimento finalizado... amostrando");
        //CO2SUM = 0.0;
        LastCO2Sampling = millis();
        co2estado = CO2SAMPINI;
      }
      estado = WAIT4FRAME;
      break;
    //
.....................................................................................
.........
    case CO2SAMPINI:
      if (CO2SampleCounter < CO2FilterLenght) { // Ainda não foram obtidas todas as
amostras necessárias...
        if ((millis() - LastCO2Sampling) >= CO2TimeBetweenSamples) {
```

```
          CO2SUM[CO2SampleCounter] = (uint16_t) CO2ppm(MG_PIN, CO2Curve);
          CO2SampleCounter++;
          LastCO2Sampling = millis();
        }
      }
      else { // Já estão todas as amostras adquiridas...
        Serial.print("Numero de amostras = ");
        Serial.println(CO2SampleCounter);
        CO2SampleCounter = 0;   // Reinicia contador de amostras
        co2estado = CO2HEAT_OF; // Manda desligar o sensor
      }
      estado = WAIT4FRAME;
      break;
    //
.....................................................................................
.........
    case CO2HEAT_OF:
      Serial.println("Desliga sensor");
      digitalWrite(CO2Vcc, HIGH);   // Desliga fonte de alimentação do sensor de CO2
      digitalWrite(LED, HIGH);      // Desliga LED
      co2estado = CO2AUPDATE;
      estado = WAIT4FRAME;
      break;
    //
.....................................................................................
.........
    case CO2AUPDATE:
      //REGS[0] = (uint16_t)(CO2SUM / (float)CO2FilterLenght); // Coloca valor no
registo
      //Serial.print("CO2 (ppm) = ");
      //Serial.println(REGS[0]);
      CO2filtrado = 0.0;
      for (int i = 0; i < CO2FilterLenght; i++) {
        CO2filtrado += float(CO2SUM[i]);
      }
      Serial.print ("Valor Filtrado = ");
      Serial.println(CO2filtrado / ((float)CO2FilterLenght));
      co2estado = CO2CHKTIME;
      estado = WAIT4FRAME;
      break;
    //
.....................................................................................
.........
    case CO2SPERROR:
      break;
  }
}
```

7.2. União das três máquinas de estados:

```
#include <SoftwareSerial.h>
#include <TinyRS485.h>
#include <TinyModbus.h>
#include "SEN0159.h"
#include <EEPROM.h> // Utilizado no futuro para guardar novo ID do slave e parâmetros
dos sensores...
#include "DFRobot_SHT20.h"
#include <Wire.h>

// Termos associados à medição do CO2
// MG_PIN - Sensor de CO2 ligado à porta A/D A0
//#define CO2Vcc                PD7        // Fonte de alimentação para o sensor de CO2
(Ativa baixa).
//#define CO2MeasurementUpdate  1*3600000  // Período de amostragem para a medição do
CO2 (em horas convertidas para millisegundos)
//#define CO2HeatingTime        5*60000    // Tempo necessário para pré-aquecer o sensor
//#define CO2TimeBetweenSamples 1*1000     // Tempo utilizado entre amostras
consecutivas do CO2 (filtragem)
//#define CO2FilterLenght       1*60       // Numero de amostras necessárias para filtro
de média
//#define LED                   PD6        // LED ligado ao pino 12 do ATMEGA328 (PD6)
(ativa-baixa)

// Para testes    |------|---|---|------|------|------|
```

```
// tempo (min)    0      1  1.5  2      3      4      5
// Duty cycle 1.5/5 = 30%

#define CO2Vcc                PD7        // Fonte de alimentação para o sensor de CO2
(Ativa baixa).
#define CO2MeasurementUpdate  300000     // Período de amostragem para a medição do CO2
(5 minutos)
#define CO2HeatingTime        60000      // Tempo necessário para pré-aquecer o sensor
(1 minuto)
#define CO2TimeBetweenSamples 500        // Tempo utilizado entre amostras consecutivas
(1/2 segundo)
#define CO2FilterLenght       1*60       // Numero de amostras necessárias para filtro
de média

#define LED                   PD6        // LED ligado ao pino 12 do ATMEGA328 (PD6)
(ativa-baixa)

// Termos associados à comunicação ModBus
#define MAX485_DE 4
#define MAX485_RE 5
#define MAX485_RO 2
#define MAX485_DI 3
//
#define TIMEOUT 100
#define FRAMELGTH 8
#define MYID 69
#define LIMITADD 0x000E

// Termos associados à medição da temperatura e humidade
#define SHT20MeasurementUpdate 60000     // Período de amostragem para a medição da
temperatura e humidade (1 minuto)

// Variáveis associadas ao SHT20
//
uint16_t REGS_STC20[3] = {0, 0, 0}; // Registo associado às observações
unsigned long LastSHT20Measurement = 0; // Retem o tempo em que foi realizada a última
medição do SHT20
float humidade    = 0.0;
float temperatura = 0.0;

// Variáveis associadas ao CO2
uint16_t REGS_CO2[3] = {0, 0, 0}; // Registo associado às observações
unsigned long myTimeCO2, MyTimeMB;
unsigned long LastCO2Measurement = 0; // Retem o tempo em que foi realizada a última
medição do CO2
unsigned long LastHeaterTime = 0;     // Retem o tempo para aquecimento do sensor
unsigned long LastCO2Sampling = 0;    // Retem o tempo em que foi realizada a última
amostragem do CO2
int CO2SampleCounter = 0;             // Contador para manter registo das amostras
adquiridas.
float CO2filtrado = 0.0;                  // Retém valor relativo à soma das amostras
do CO2
uint16_t CO2SUM[CO2FilterLenght];

// Variáveis associadas ao ModBus
int bufcnt = 0;
const int BUFFER_SIZE = 10;
uint8_t   RX[BUFFER_SIZE] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; // Buffer com a mensagem
recebida do master
uint8_t   TX[BUFFER_SIZE] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; // Buffer com a mensagem de
retorno para o master
int       TxLen = 0;                            // Dimensão da mensagem a enviar
uint16_t  CRC = 0x0000;                         // CRC da mensagem a enviar
uint16_t  BaseAddress = 0x0000;                 // Endereço base solicitado pelo Master
uint16_t  OfftAddress = 0x0000;                 // Offser associado ao endereço
solicitado pelo Master

int i = 0; // Índice genérico usado em ciclos FOR

SoftwareSerial SoftSerial(MAX485_RO, MAX485_DI); // RX, TX
TinyRS485 RS485;
DFRobot_SHT20    sht20;
// ------------------------------------------------------------------------------------
----------------------------
// MÁQUINA DE ESTADOS
// ------------------------------------------------------------------------------------
----------------------------
```

```
enum estadosistema {
  SETRCVMODE, // Coloca o MAX485 no modo de receção
  WAIT4FRAME, // Agurada que uma frame completa seja recebida
  TSTINFRAME, // Verifica se há algum erro com a frame
  MESSAGE4ME, // A mensagem é para mim. Verifica se possuo a função solicitada
  RDFUNCTION,
  WEFUNCTION,
  FUNCNERROR,
  ADDREERROR,
  DECODE4RDE, // Descodifica para envio de dados
  DECODE4WTE, // Descodifica para alteração de variáveis
  ANSWERFRME, // Responde a um pedido
  ERRORFRAME, // Responde com um erro
  CO2CHKTIME, // Verifica se é altura para iniciar novo sequencia de amostragem
  CO2HEAT_ON, // Liga sensor para iniciar processo de aquecimento
  CO2HEAT_OF, // Desliga sensor
  CO2HEAT_OK, // Tempo de aquecimento completo
  CO2SAMPINI, // Inicia processo de amostragem
  CO2AUPDATE, // Atualiza registos e variáveis de estado
  CO2SPERROR, // Erro na amostragem
  SHT20CKTME, //
  SHT20GTDTA  // STH20 get data
};

estadosistema sht20estado = SHT20CKTME;
estadosistema co2estado   = CO2CHKTIME;
estadosistema estado      = SETRCVMODE;

// ----------------------------------------------------------------------------------------
----------------------------
// bool checkForFrame(void)
// ----------------------------------------------------------------------------------------
----------------------------
bool checkForFrame()
{
  bufcnt = 0;
  if (RS485.available())
  {
    MyTimeMB = millis();
    RX[bufcnt] = RS485.receive();
    bufcnt++;
    // Baudrate = 115200 bps, 80 bits => 0.7 ms para receber a frame Modbus
    while ((millis() -  MyTimeMB) < TIMEOUT)
    {
      if (RS485.available())
      {
        RX[bufcnt] = RS485.receive();
        bufcnt++;
      }
    }
  }
  // O tempo terminou desde que foi recebido o primeiro caracter... verificar se a frame
completa foi recebida
  if (bufcnt >= FRAMELGTH) return (true);
  else return (false);
}

// ----------------------------------------------------------------------------------------
----------------------------
// void setup(void)
// ----------------------------------------------------------------------------------------
----------------------------
void setup()
{
  Serial.begin(115200);
  // ModBus
  SoftSerial.begin(115200); // Porto série (software) usado pelo transceiver RS485
  pinMode(MAX485_DE, OUTPUT);
  pinMode(MAX485_RE, OUTPUT);
  RS485.begin(SoftSerial, MAX485_DE, MAX485_RE);
  // Sensor SHT20
  sht20.initSHT20();
  delay(100);
  sht20.checkSHT20();
  // CO2
  pinMode(LED, OUTPUT);           // LED para debug ou sinalização
  pinMode(CO2Vcc, OUTPUT);        // Estabelece porto para controlo de CO2Vcc
```

```
  digitalWrite(CO2Vcc, HIGH);     // Desliga fonte de alimentação do sensor de CO2
  digitalWrite(LED, HIGH);        // Desliga fonte de alimentação do sensor de CO2
  LastCO2Measurement   = millis() - CO2MeasurementUpdate;   // Força primeira medição
  LastSHT20Measurement = millis() - SHT20MeasurementUpdate; // Força primeira medição

}

// ------------------------------------------------------------------------------------
----------------------------
// void loop(void)
// ------------------------------------------------------------------------------------
----------------------------
void loop()
{

  switch (estado)
  {
    //
.....................................................................................
...........
    // Máquina de estados ModBus
    //
.....................................................................................
...........
    case SETRCVMODE:
      while (RS485.available()) RS485.receive(); // Limpa buffer
      Serial.println("Inicio da maquina de estados");
      digitalWrite(MAX485_DE, LOW); // Desativa DRIVER
      digitalWrite(MAX485_RE, LOW); // Ativa RECEIVER
      estado = WAIT4FRAME;
      break;
    //
.....................................................................................
.........
    case WAIT4FRAME:
      if (checkForFrame()) estado = TSTINFRAME;
      else estado = sht20estado;
      break;
    //
.....................................................................................
.........
    case TSTINFRAME:
      // Mostra frame que recebeu
      for (i = 0; i < bufcnt; i++) Serial.print(RX[i], HEX);

      Serial.println("");
      Serial.print("MyID: ");

      Serial.println(RX[0], DEC);
      // Verifica integridade...
      // RX[0]^=0x01; // Simula erro de transmissão (alteração de 1 bit)
      if (compute_crc(RX, bufcnt) == 0) { // CRC OK
        // Verifica se a frame é para mim...
        if (RX[0] == MYID) {
          Serial.println("A mensagem E para mim");
          estado = MESSAGE4ME; // Sim, sou eu...
        }
        else {
          Serial.println("A mensagem NAO e para mim");
          estado = WAIT4FRAME; // Nop, aguarda por nova frame...
        }
      }
      else { // ERRO DE TRANSMISSÂO
        Serial.println("ERRO de CRC");
        estado = ERRORFRAME;
      }
      break;
    //
.....................................................................................
.........
    case MESSAGE4ME:
      // Verifica se o slave pode realizar a função solicitada em RX[1]
      switch (RX[1]) {
        case 0x03:  estado = RDFUNCTION;
          Serial.println("Pedem para enviar qualquer coisa");
          break;
        case 0x06:  estado = WEFUNCTION;
```

```
          Serial.println("Pedem para alterar qualquer coisa");
          break;
        default:    estado = FUNCNERROR;
          Serial.println("Nao sei o que querem");
          break;
      }
      break;
    //
.....................................................................................
.........
    case RDFUNCTION:
      // Verifica se o endereço está dentro dos limites solicitados
      BaseAddress = (((uint16_t)RX[2]) << 8) + ((uint16_t)RX[3]); // Calcula endereço
base
      OfftAddress = (((uint16_t)RX[4]) << 8) + ((uint16_t)RX[5]); // Calcula offset
      if (BaseAddress + OfftAddress > LIMITADD) {
        estado = ADDREERROR;
        Serial.println("Endereço de leitura fora dos limites");
      }
      else {
        estado = DECODE4RDE; // Descodifica pedido de leitura
      }
      break;
    //
.....................................................................................
.........
    case WEFUNCTION:
      // Verifica se o endereço está dentro dos limites solicitados
      BaseAddress = (((uint16_t)RX[2]) << 8) + ((uint16_t)RX[3]); // Calcula endereço
base
      OfftAddress = (((uint16_t)RX[4]) << 8) + ((uint16_t)RX[5]); // Calcula offset
      if (BaseAddress + OfftAddress > LIMITADD) {
        estado = ADDREERROR;
        Serial.println("Endereco de escrita fora dos limites");
      }
      else {
        estado = DECODE4WTE; // Descodifica pedido de escrita
      }
      break;
    //
.....................................................................................
.........
    case FUNCNERROR:
      // Constroi frame
      Serial.println("Retorna info de Function Error");
      TX[0] = MYID;
      TX[1] = RX[1] + 0x80;
      TX[2] = 0x01;
      TxLen = 3;
      estado = ANSWERFRME;
      break;
    //
.....................................................................................
.........
    case ADDREERROR:
      // Constroi frame
      Serial.println("Retorna info de Address Error");
      TX[0] = MYID;
      TX[1] = RX[1] + 0x80;
      TX[2] = 0x02;
      TxLen = 3;
      estado = ANSWERFRME;
      break;
    //
.....................................................................................
.........
    case DECODE4RDE:
      Serial.println("Descodifico frame e executo pedido");
      // Coleta a informação necessária para originar a resposta
      // Constroi frame
      TX[0] = MYID;
      TX[1] = RX[1];
      TX[2] = 0x02;
      TX[3] = 0xAB;
      TX[4] = 0xCD;
      TxLen = 5;
      // Rotina para execução do pedido.
```

```
      estado = ANSWERFRME;
      break;
    //
.....................................................................................
.........
    case ANSWERFRME:
      Serial.println("Resposta a Frame (via RS485)");
      // Adiciona CRC à frame
      CRC   = compute_crc(TX, TxLen);
      TX[TxLen++] = (uint8_t) CRC;
      TX[TxLen++] = (uint8_t) (CRC >> 8);
      // Ativa MAX485 para TX
      digitalWrite(MAX485_DE, HIGH); // Ativa DRIVER
      digitalWrite(MAX485_RE, HIGH); // Desativa RECEIVER
      // Transmite dados...
      for (i = 0; i < TxLen; i++) RS485.transmit(TX[i]);
      estado = SETRCVMODE;
      break;
    //
.....................................................................................
.........
    case ERRORFRAME:
      Serial.println("ERRO: CRC inválido");
      TX[0] = MYID;
      TX[1] = RX[1] + 0x80;
      TX[2] = 0x03;
      TxLen = 3;
      estado = ANSWERFRME;
      break;
    //
.....................................................................................
...........
    // Máquina de estados CO2
    //
.....................................................................................
...........
    case CO2CHKTIME:
      if ((millis() - LastCO2Measurement) >= CO2MeasurementUpdate) {
        LastCO2Measurement = millis();
        co2estado = CO2HEAT_ON;
      }
      estado = WAIT4FRAME;
      break;
    //
.....................................................................................
.........
    case CO2HEAT_ON:
      Serial.println("Inicia aquecimento do sensor");
      digitalWrite(CO2Vcc, LOW);   // Liga fonte de alimentação do sensor de CO2
      digitalWrite(LED, LOW);      // Liga LED
      LastHeaterTime = millis();
      co2estado = CO2HEAT_OK;
      estado = WAIT4FRAME;
      break;
    //
.....................................................................................
.........
    case CO2HEAT_OK:
      if ((millis() - LastHeaterTime) >= CO2HeatingTime) {
        Serial.println("Aquecimento finalizado... amostrando");
        //CO2SUM = 0.0;
        LastCO2Sampling = millis();
        co2estado = CO2SAMPINI;
      }
      estado = WAIT4FRAME;
      break;
    //
.....................................................................................
.........
    case CO2SAMPINI:
      if (CO2SampleCounter < CO2FilterLenght) { // Ainda não foram obtidas todas as
amostras necessárias...
        if ((millis() - LastCO2Sampling) >= CO2TimeBetweenSamples) {
          CO2SUM[CO2SampleCounter] = (uint16_t) CO2ppm(MG_PIN, CO2Curve);
          CO2SampleCounter++;
          LastCO2Sampling = millis();
        }
```

```
      }
      else { // Já estão todas as amostras adquiridas...
        Serial.print("Numero de amostras = ");
        Serial.println(CO2SampleCounter);
        CO2SampleCounter = 0;   // Reinicia contador de amostras
        co2estado = CO2HEAT_OF; // Manda desligar o sensor
      }
      estado = WAIT4FRAME;
      break;
    //
.....................................................................................
.........
    case CO2HEAT_OF:
      Serial.println("Desliga sensor");
      digitalWrite(CO2Vcc, HIGH);   // Desliga fonte de alimentação do sensor de CO2
      digitalWrite(LED, HIGH);      // Desliga LED
      co2estado = CO2AUPDATE;
      estado = WAIT4FRAME;
      break;
    //
.....................................................................................
.........
    case CO2AUPDATE:
      //REGS[0] = (uint16_t)(CO2SUM / (float)CO2FilterLenght); // Coloca valor no
registo
      //Serial.print("CO2 (ppm) = ");
      //Serial.println(REGS[0]);
      CO2filtrado = 0.0;
      for (int i = 0; i < CO2FilterLenght; i++) {
        CO2filtrado += float(CO2SUM[i]);
      }
      Serial.print("Timetag:");                         // Debug
      Serial.print(millis());
      Serial.print(" CO2 = ");
      Serial.print(CO2filtrado / ((float)CO2FilterLenght));
      Serial.println(" ppm");
      co2estado = CO2CHKTIME;
      estado = WAIT4FRAME;
      break;
    //
.....................................................................................
.........
    case CO2SPERROR:
      break;
    //
.....................................................................................
...........
    // Máquina de estados SHT20
    //
.....................................................................................
...........
    case SHT20CKTME:
      if ((millis() - LastSHT20Measurement) >= SHT20MeasurementUpdate) {
        LastSHT20Measurement = millis();
        sht20estado = SHT20GTDTA;
      }
      estado = co2estado;
      break;
    //
.....................................................................................
.........
    case SHT20GTDTA:
      humidade = sht20.readHumidity();                  // Observa humidade do ar
      temperatura = sht20.readTemperature();            // Observa temperaura do ar
      Serial.print("Timetag:");                         // Debug
      Serial.print(millis());
      Serial.print(" T = ");
      Serial.print(temperatura, 1);
      Serial.print("ºC");
      Serial.print(" RH = ");
      Serial.print(humidade, 1);
      Serial.println("%");
      sht20estado = SHT20CKTME;
      estado = co2estado;
      break;
  }
```

```
}
```

7.3. Versão 3.0

Existe um problema (crónico) na utilização da biblioteca Software Serial com o RS485. Na realidade, após um número exaustivo de ensaios, observou-se que a comunicação com o RS485 dongle via porto série virtual levava a uma taxa de erro de, aproximadamente, 30% o que é inaceitável. A solução passou pela utilização do porto série em hardware do ATMega328. Em particular, este porto passou a estar associado à comunicação RS485 e o porto série por software para debug via RS232. O código fonte encontra-se no repositório do github e apresenta-se por conveniência, na listagem a seguir.

```
#include <SoftwareSerial.h>
#include <TinyRS485.h>
#include <TinyModbus.h>
#include "SEN0159.h"
#include <EEPROM.h> // Utilizado no futuro para guardar novo ID do slave e parâmetros
dos sensores...
#include "DFRobot_SHT20.h"
#include <Wire.h>

// Termos associados à medição do CO2
// MG_PIN - Sensor de CO2 ligado à porta A/D A0
//#define CO2Vcc                PD7        // Fonte de alimentação para o sensor de CO2
(Ativa baixa).
//#define CO2MeasurementUpdate  1*3600000  // Período de amostragem para a medição do
CO2 (em horas convertidas para millisegundos)
//#define CO2HeatingTime        5*60000    // Tempo necessário para pré-aquecer o sensor
//#define CO2TimeBetweenSamples 1*1000     // Tempo utilizado entre amostras
consecutivas do CO2 (filtragem)
//#define CO2FilterLenght       1*60       // Numero de amostras necessárias para filtro
de média
//#define LED                   PD6        // LED ligado ao pino 12 do ATMEGA328 (PD6)
(ativa-baixa)

// Para testes    |------|---|---|------|------|------|
// tempo (min)    0      1  1.5  2      3      4      5
// Duty cycle 1.5/5 = 30%

#define CO2Vcc                PD7        // Fonte de alimentação para o sensor de CO2
(Ativa baixa).
#define CO2MeasurementUpdate  300000     // Período de amostragem para a medição do CO2
(5 minutos)
#define CO2HeatingTime        60000      // Tempo necessário para pré-aquecer o sensor
(1 minuto)
#define CO2TimeBetweenSamples 500        // Tempo utilizado entre amostras consecutivas
(1/2 segundo)
#define CO2FilterLenght       100        // Numero de amostras necessárias para filtro
de média

#define LED                   PD6        // LED ligado ao pino 12 do ATMEGA328 (PD6)
(ativa-baixa)

// Termos associados à comunicação ModBus
#define MAX485_DE 4
#define MAX485_RE 5
#define MAX485_RO 2
#define MAX485_DI 3
//
#define TIMEOUT 100
#define FRAMELGTH 8
#define MYID 69
```

```
#define OFSTADDLIM 0x0003
#define BASEADDLIM 0x0008
#define LIMIT_ADDR OFSTADDLIM+BASEADDLIM

// Termos associados à medição da temperatura e humidade
#define SHT20MeasurementUpdate 60000     // Período de amostragem para a medição da
temperatura e humidade (1 minuto)

// Variáveis associadas ao SHT20
//
unsigned long LastSHT20Measurement = 0; // Retem o tempo em que foi realizada a última
medição do SHT20
float humidade    = 0.0;
float temperatura = 0.0;

// Variáveis associadas ao CO2
unsigned long myTimeCO2, MyTimeMB;
unsigned long LastCO2Measurement = 0; // Retem o tempo em que foi realizada a última
medição do CO2
unsigned long LastHeaterTime = 0;     // Retem o tempo para aquecimento do sensor
unsigned long LastCO2Sampling = 0;    // Retem o tempo em que foi realizada a última
amostragem do CO2
int CO2SampleCounter = 0;             // Contador para manter registo das amostras
adquiridas.
float CO2filtrado = 0.0;                   // Retém valor relativo à soma das amostras
do CO2
uint16_t CO2SUM[CO2FilterLenght];
//           CO2 (350 ppm) Temp 19.7 Humidade 58.0%
//uint16_t REGS[3] = {0x015E, 0x4CF4, 0x16AB}; // Registo associado às observações
uint16_t REGS[3] = {0x015E, 0xFC7C, 0x16AB}; // Registo associado às observações

// Variáveis associadas ao ModBus
int bufcnt = 0;
const int BUFFER_SIZE = 10;
uint8_t   RX[BUFFER_SIZE] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; // Buffer com a mensagem
recebida do master
uint8_t   TX[BUFFER_SIZE] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; // Buffer com a mensagem de
retorno para o master
int       TxLen = 0;                            // Dimensão da mensagem a enviar
uint16_t  CRC = 0x0000;                         // CRC da mensagem a enviar
uint16_t  BaseAddress = 0x0000;                 // Endereço base solicitado pelo Master
uint16_t  OfftAddress = 0x0000;                 // Offser associado ao endereço
solicitado pelo Master

int i = 0; // Índice genérico usado em ciclos FOR

SoftwareSerial SoftSerial(MAX485_RO, MAX485_DI); // RX, TX
TinyRS485 RS485;
DFRobot_SHT20    sht20;
// ------------------------------------------------------------------------------------
----------------------------
// MÁQUINA DE ESTADOS
// ------------------------------------------------------------------------------------
----------------------------
enum estadosistema {
  SETRCVMODE, // Coloca o MAX485 no modo de receção
  WAIT4FRAME, // Agurada que uma frame completa seja recebida
  TSTINFRAME, // Verifica se há algum erro com a frame
  MESSAGE4ME, // A mensagem é para mim. Verifica se possuo a função solicitada
  RDFUNCTION,
  WEFUNCTION,
  FUNCNERROR,
  ADDREERROR,
  DECODE4RDE, // Descodifica para envio de dados
  DECODE4WTE, // Descodifica para alteração de variáveis
  ANSWERFRME, // Responde a um pedido
  ERRORFRAME, // Responde com um erro
  CO2CHKTIME, // Verifica se é altura para iniciar novo sequencia de amostragem
  CO2HEAT_ON, // Liga sensor para iniciar processo de aquecimento
  CO2HEAT_OF, // Desliga sensor
  CO2HEAT_OK, // Tempo de aquecimento completo
  CO2SAMPINI, // Inicia processo de amostragem
  CO2AUPDATE, // Atualiza registos e variáveis de estado
  CO2SPERROR, // Erro na amostragem
  SHT20CKTME, //
  SHT20GTDTA  // STH20 get data
};
```

```
estadosistema sht20estado = SHT20CKTME;
estadosistema co2estado   = CO2CHKTIME;
estadosistema estado      = SETRCVMODE;

// ------------------------------------------------------------------------------------
----------------------------
// bool checkForFrame(void)
// ------------------------------------------------------------------------------------
----------------------------
bool checkForFrame()
{
  bufcnt = 0;
  if (RS485.available())
  {
    MyTimeMB = millis();
//    RX[bufcnt] = RS485.receive();
//    bufcnt++;
    // Baudrate = 115200 bps, 80 bits => 0.7 ms para receber a frame Modbus
    while ((millis() -  MyTimeMB) < TIMEOUT)
    {
      if (RS485.available())
      {
        RX[bufcnt] = RS485.receive();
        bufcnt++;
      }
    }
  }
  // O tempo terminou desde que foi recebido o primeiro caracter... verificar se a frame
completa foi recebida
  if (bufcnt >= FRAMELGTH) return (true);
  else return (false);
}

// ------------------------------------------------------------------------------------
----------------------------
// void setup(void)
// ------------------------------------------------------------------------------------
----------------------------
void setup()
{
  Serial.begin(115200);
  // ModBus
  SoftSerial.begin(115200); // Porto série (software) usado pelo transceiver RS485
  pinMode(MAX485_DE, OUTPUT);
  pinMode(MAX485_RE, OUTPUT);
  RS485.begin(Serial, MAX485_DE, MAX485_RE);
  // Sensor SHT20
  sht20.initSHT20();
  delay(100);
  sht20.checkSHT20();
  // CO2
  pinMode(LED, OUTPUT);           // LED para debug ou sinalização
  pinMode(CO2Vcc, OUTPUT);        // Estabelece porto para controlo de CO2Vcc
  digitalWrite(CO2Vcc, HIGH);     // Desliga fonte de alimentação do sensor de CO2
  digitalWrite(LED, HIGH);        // Desliga fonte de alimentação do sensor de CO2
  LastCO2Measurement   = millis() - CO2MeasurementUpdate;   // Força primeira medição
  LastSHT20Measurement = millis() - SHT20MeasurementUpdate; // Força primeira medição

}

// ------------------------------------------------------------------------------------
----------------------------
// void loop(void)
// ------------------------------------------------------------------------------------
----------------------------
void loop()
{

  switch (estado)
  {
    //
.....................................................................................
...........
    // Máquina de estados ModBus
```

```
    //
.....................................................................................
...........
    case SETRCVMODE:
      while (RS485.available()) RS485.receive(); // Limpa buffer
      SoftSerial.println("Inicio da maquina de estados");
      digitalWrite(MAX485_DE, LOW); // Desativa DRIVER
      digitalWrite(MAX485_RE, LOW); // Ativa RECEIVER
      estado = WAIT4FRAME;
      break;
    //
.....................................................................................
.........
    case WAIT4FRAME:
      if (checkForFrame()) estado = TSTINFRAME;
      else estado = sht20estado;
      break;
    //
.....................................................................................
.........
    case TSTINFRAME:
      // Mostra frame que recebeu
      for (i = 0; i < bufcnt; i++) Serial.print(RX[i], HEX);

      SoftSerial.println("");
      SoftSerial.print("MyID: ");

      SoftSerial.println(RX[0], DEC);
      // Verifica integridade...
      // RX[0]^=0x01; // Simula erro de transmissão (alteração de 1 bit)
      if (compute_crc(RX, bufcnt) == 0) { // CRC OK
        // Verifica se a frame é para mim...
        if (RX[0] == MYID) {
          SoftSerial.println("A mensagem E para mim");
          estado = MESSAGE4ME; // Sim, sou eu...
        }
        else {
          SoftSerial.println("A mensagem NAO e para mim");
          estado = WAIT4FRAME; // Nop, aguarda por nova frame...
        }
      }
      else { // ERRO DE TRANSMISSÂO
        SoftSerial.println("ERRO de CRC");
        estado = ERRORFRAME;
      }
      break;
    //
.....................................................................................
.........
    case MESSAGE4ME:
      // Verifica se o slave pode realizar a função solicitada em RX[1]
      switch (RX[1]) {
        case 0x03:  estado = RDFUNCTION;
          SoftSerial.println("Pedem para enviar qualquer coisa");
          break;
        case 0x06:  estado = WEFUNCTION;
          SoftSerial.println("Pedem para alterar qualquer coisa");
          break;
        default:    estado = FUNCNERROR;
          SoftSerial.println("Nao sei o que querem");
          break;
      }
      break;
    //
.....................................................................................
.........
    case RDFUNCTION:
      // Verifica se o endereço está dentro dos limites solicitados
      BaseAddress = (((uint16_t)RX[2]) << 8) + ((uint16_t)RX[3]); // Calcula endereço
base
      OfftAddress = (((uint16_t)RX[4]) << 8) + ((uint16_t)RX[5]); // Calcula offset
      SoftSerial.print("Endereço base: ");
      SoftSerial.println(BaseAddress);
      SoftSerial.print("Offset: ");
      SoftSerial.println(OfftAddress);

      if (BaseAddress<BASEADDLIM || BaseAddress + OfftAddress > LIMIT_ADDR) {
```

```
        estado = ADDREERROR;
        SoftSerial.println("Endereço de leitura fora dos limites");
      }
      else {
        estado = DECODE4RDE; // Descodifica pedido de leitura
      }
      break;
    //
.....................................................................................
.........
    case WEFUNCTION:
      // Verifica se o endereço está dentro dos limites solicitados
      BaseAddress = (((uint16_t)RX[2]) << 8) + ((uint16_t)RX[3]); // Calcula endereço
base
      OfftAddress = (((uint16_t)RX[4]) << 8) + ((uint16_t)RX[5]); // Calcula offset
      if (BaseAddress<BASEADDLIM || BaseAddress + OfftAddress > LIMIT_ADDR) {
        estado = ADDREERROR;
        SoftSerial.println("Endereco de escrita fora dos limites");
      }
      else {
        estado = DECODE4WTE; // Descodifica pedido de escrita
      }
      break;
    //
.....................................................................................
.........
    case FUNCNERROR:
      // Constroi frame
      SoftSerial.println("Retorna info de Function Error");
      TX[0] = MYID;
      TX[1] = RX[1] + 0x80;
      TX[2] = 0x01;
      TxLen = 3;
      estado = ANSWERFRME;
      break;
    //
.....................................................................................
.........
    case ADDREERROR:
      // Constroi frame
      SoftSerial.println("Retorna info de Address Error");
      TX[0] = MYID;
      TX[1] = RX[1] + 0x80;
      TX[2] = 0x02;
      TxLen = 3;
      estado = ANSWERFRME;
      break;
    //
.....................................................................................
.........
    case DECODE4RDE:
      SoftSerial.println("Descodifico frame e executo pedido");
      // Coleta a informação necessária para originar a resposta

      // Constroi frame
      TX[0] = MYID;
      TX[1] = RX[1];
      TX[2] = 0x02*OfftAddress;
      for (i=0;i<OfftAddress;i++){
        TX[3+2*i] = (uint8_t)(REGS[i]>>8);
        TX[3+2*i+1] = (uint8_t) REGS[i];
      }
      TxLen = 3+2*OfftAddress;
      // Rotina para execução do pedido.
      estado = ANSWERFRME;
      break;
    //
.....................................................................................
.........
    case ANSWERFRME:
      SoftSerial.println("Resposta a Frame (via RS485)");
      // Adiciona CRC à frame
      CRC   = compute_crc(TX, TxLen);
      TX[TxLen++] = (uint8_t) CRC;
      TX[TxLen++] = (uint8_t) (CRC >> 8);
      // Ativa MAX485 para TX
      digitalWrite(MAX485_DE, HIGH); // Ativa DRIVER
```

```
      digitalWrite(MAX485_RE, HIGH); // Desativa RECEIVER
      // Transmite dados...
      for (i = 0; i < TxLen; i++) RS485.transmit(TX[i]);
      estado = SETRCVMODE;
      break;
    //
.....................................................................................
.........
    case ERRORFRAME:
      SoftSerial.println("ERRO: CRC inválido");
      TX[0] = MYID;
      TX[1] = RX[1] + 0x80;
      TX[2] = 0x03;
      TxLen = 3;
      estado = ANSWERFRME;
      break;
    //
.....................................................................................
...........
    // Máquina de estados CO2
    //
.....................................................................................
...........
    case CO2CHKTIME:
      if ((millis() - LastCO2Measurement) >= CO2MeasurementUpdate) {
        LastCO2Measurement = millis();
        co2estado = CO2HEAT_ON;
      }
      estado = WAIT4FRAME;
      break;
    //
.....................................................................................
.........
    case CO2HEAT_ON:
      SoftSerial.println("Inicia aquecimento do sensor");
      digitalWrite(CO2Vcc, LOW);   // Liga fonte de alimentação do sensor de CO2
      digitalWrite(LED, LOW);      // Liga LED
      LastHeaterTime = millis();
      co2estado = CO2HEAT_OK;
      estado = WAIT4FRAME;
      break;
    //
.....................................................................................
.........
    case CO2HEAT_OK:
      if ((millis() - LastHeaterTime) >= CO2HeatingTime) {
        SoftSerial.println("Aquecimento finalizado... amostrando");
        //CO2SUM = 0.0;
        LastCO2Sampling = millis();
        co2estado = CO2SAMPINI;
      }
      estado = WAIT4FRAME;
      break;
    //
.....................................................................................
.........
    case CO2SAMPINI:
      if (CO2SampleCounter < CO2FilterLenght) { // Ainda não foram obtidas todas as
amostras necessárias...
        if ((millis() - LastCO2Sampling) >= CO2TimeBetweenSamples) {
          CO2SUM[CO2SampleCounter] = (uint16_t) CO2ppm(MG_PIN, CO2Curve);
          CO2SampleCounter++;
          LastCO2Sampling = millis();
        }
      }
      else { // Já estão todas as amostras adquiridas...
        SoftSerial.print("Numero de amostras = ");
        SoftSerial.println(CO2SampleCounter);
        CO2SampleCounter = 0;   // Reinicia contador de amostras
        co2estado = CO2HEAT_OF; // Manda desligar o sensor
      }
      estado = WAIT4FRAME;
      break;
    //
.....................................................................................
.........
    case CO2HEAT_OF:
```

```
      SoftSerial.println("Desliga sensor");
      digitalWrite(CO2Vcc, HIGH);   // Desliga fonte de alimentação do sensor de CO2
      digitalWrite(LED, HIGH);      // Desliga LED
      co2estado = CO2AUPDATE;
      estado = WAIT4FRAME;
      break;
    //
.....................................................................................
.........
    case CO2AUPDATE:
      CO2filtrado = 0.0;
      for (int i = 0; i < CO2FilterLenght; i++) {
        CO2filtrado += float(CO2SUM[i]);
      }
      SoftSerial.print("Timetag:");                          // Debug
      SoftSerial.print(millis());
      SoftSerial.print(" CO2 = ");
      SoftSerial.print(CO2filtrado / ((float)CO2FilterLenght));
      SoftSerial.println(" ppm");
      REGS[0] = (uint16_t) (10.0 * CO2filtrado / ((float)CO2FilterLenght));
      co2estado = CO2CHKTIME;
      estado = WAIT4FRAME;
      break;
    //
.....................................................................................
.........
    case CO2SPERROR:
      break;
    //
.....................................................................................
...........
    // Máquina de estados SHT20
    //
.....................................................................................
...........
    case SHT20CKTME:
      if ((millis() - LastSHT20Measurement) >= SHT20MeasurementUpdate) {
        LastSHT20Measurement = millis();
        sht20estado = SHT20GTDTA;
      }
      estado = co2estado;
      break;
    //
.....................................................................................
.........
    case SHT20GTDTA:
      humidade = sht20.readHumidity();                  // Observa humidade do ar
      temperatura = sht20.readTemperature();            // Observa temperaura do ar
      REGS[1] = (uint16_t) (100*temperatura);
      REGS[2] = (uint16_t) (100*humidade);
      SoftSerial.print("Timetag:");                          // Debug
      SoftSerial.print(millis());
      SoftSerial.print(" T = ");
      SoftSerial.print(temperatura, 1);
      SoftSerial.print("ºC");
      SoftSerial.print(" RH = ");
      SoftSerial.print(humidade, 1);
      SoftSerial.println("%");
      sht20estado = SHT20CKTME;
      estado = co2estado;
      break;
  }
}
```