

Universidade Federal de Uberlândia
Faculdade de Engenharia Elétrica - FEELT
Pró- Reitoria de Pesquisa e Pós-graduação - PROPP

Thiago Fernando Cuevas Mestanza

Relatório Parcial de Iniciação Científica
Desenvolvimento e Implementação de um Gateway Comunicador entre Protocolos
Modbus TCP/RTU

Uberlândia, MG

2022

Thiago Fernando Cuevas Mestanza

**Desenvolvimento e Implementação de um Gateway Comunicador entre Protocolos
Modbus TCP/RTU**

Relatório parcial do projeto de Iniciação Científica voltado para o Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Uberlândia. Edital DIRPE Nº 10/2021 - PIBIC/FAPEMIG/UFU.

Orientador: Prof. Dr. Renato F. Fernandes Jr.

Uberlândia, MG

2022

RESUMO

A constante busca pela otimização de processos tem sido um dos principais estímulos para os avanços técnico-científicos. O campo industrial é um dos setores que mais se nutre desses progressos, ficando evidente no âmbito das redes industriais, peça fundamental hoje para a construção da indústria moderna. Nesse sentido, a efervescente evolução desse setor faz com que as tecnologias desenvolvidas, rapidamente, substituam umas às outras, tornando algumas obsoletas ou incomunicáveis quando comparadas com os equipamentos mais modernos. Dessa forma, este trabalho tem como objetivo desenvolver um Gateway Modbus capaz de servir como alternativa de baixo custo para reutilização de Controladores Lógicos Programáveis (CLP's) antigos, que possuem barramentos diferentes dos modernos, no intuito de prolongar a vida útil desses equipamentos e dar a possibilidade de expandir os limites e conexões de uma rede industrial convencional. O Gateway será desenvolvido usando a placa BeagleBone Black para atuar como intercessor dos protocolos Modbus TCP/RTU, e assim ser capaz de efetivar a comunicação dos controladores que possuem barramento RS232 e RS485, com os que possuem conexão Ethernet/IP. Até o momento, através dos testes de firmwares já desenvolvidos, a pesquisa logrou a validação de equipamentos, softwares de simulação, linguagens e métodos de programação, que serão detalhadas neste trabalho.

1. INTRODUÇÃO

Atualmente as redes baseadas em ethernet tem dominado o setor industrial frente a outras redes com barramentos de campo já consolidados, e que tiveram sua concepção no início do século. Esta transição para a ethernet industrial é muito impulsionada pela evolução dos equipamentos de rede ethernet, pela integração entre os diferentes níveis de automação dentro da indústria, e também pelas tecnologias de IoT (*Internet of Things*) e Industria 4.0 [1].

Nesse contexto, as redes industriais atuais interligam as várias camadas do processo industrial, desde o nível mais baixo, chamado de nível de chão de fábrica, onde estão localizados os sensores e atuadores responsáveis pelo andamento do processo produtivo, até o nível mais alto de controle central. Através das sub redes, todos os equipamentos da rede podem comunicar-se, mesmo estando distantes a dezenas de quilômetros [2]. A informação que percorre o sistema de comunicação desde os sensores até os controladores, é dada em protocolos.

Entre os vários protocolos disponíveis no mercado, um dos mais usados é o Modbus, que apesar de antigo no campo das redes industriais, é talvez o protocolo de mais larga utilização, já que diversos controladores e ferramentas para desenvolvimento de sistemas supervisórios o utilizam devido a sua facilidade de implementação e baixo custo [3]. O protocolo Modbus ganhou bastante adesão com o padrão de comunicação serial, RS232 e posteriormente RS485, configurando o Modbus RTU (*Remote Terminal Unit*). Porém, com o advento e popularização das redes Ethernet e de sistemas de comunicação mais velozes, o protocolo Modbus foi atualizado para suportar o meio físico Ethernet, dando origem ao Modbus TCP (*Transmission Control Protocol*). Entretanto o pacote de dados desse protocolo não é o mesmo do protocolo Modbus RTU, tornando necessária uma conversão quando se pretende usar os dois protocolos. Essa conversão pode ser dada por meio dos *Gateways*.

O conceito de *Gateway* é um dos mais importantes dentro da lógica de IoT, já que consiste em um *proxy*, isto é, um intermediador, entre a rede de detecção e as camadas de aplicação, estabelecendo a comunicação entre sensores, e as camadas superiores da rede [4]. No ambiente industrial, *Gateways* cumprem a função de intercessores de comunicação remota, entre sensores, atuadores, transmissores, com sistemas de controle e supervisão da planta de produção. [5].

2. OBJETIVOS

Esta pesquisa se propõe a desenvolver um *Gateway*, isto é, um intercessor de protocolos, capaz de mediar a troca de informações entre uma rede Modbus RTU, composta de equipamentos mais antigos, e uma rede Modbus TCP, composta de equipamentos mais modernos. O Gateway será desenvolvido utilizando o microcontrolador BeagleBone Black (BBB) que permitirá embarcar um sistema operacional Linux, sendo possível assim, trabalhar com aplicações mais complexas e sistemas multitarefas.

3. METODOLOGIA

Visando atingir os objetivos mencionados e tentando responder às questões surgidas durante o desenvolvimento deste projeto, foram realizadas atividades práticas e teóricas alinhadas com as atividades previstas na proposta de pesquisa. Para a realização de testes do sistema, foram utilizados dispositivos e módulos complementares que foram fornecidos pelo laboratório. Tais dispositivos estão representados na figura 1. Entre as atividades desempenhadas até então, estão:

3.1 Desenvolvimento de um Firmware básico na Beaglebone Black.

Foi realizado um reconhecimento prático do microcontrolador e do sistema operacional Linux embarcado nele. Os procedimentos foram realizados de acordo com os roteiros de Firmwares (programas em linguagem computacional), descritos nos manuais elaborados pelo grupo de pesquisa e colaboradores do Laboratório de Automação, Sistemas Eletrônicos e Controle (LASEC). Foram implementados programas básicos em linguagem C, como firmwares simples de impressão na tela, entrada e saída digital e configuração UART.

3.2 Desenvolvimento e testes de um Firmware escravo Modbus TCP.

Desenvolveu-se um Firmware, baseado na biblioteca *open-source* FreeModbus, capaz de receber e interpretar as instruções de um mestre TCP. A validação do funcionamento do firmware foi feita através de testes utilizando os softwares simuladores mestre Modbus, ModScan e QModbus. Nos testes, conectou-se, para alimentação da BBB, um cabo do tipo Micro USB (Figura 1 – F) entre a entrada de alimentação da placa e uma porta USB do computador. Para estabelecer a conexão Modbus TCP entre simulador e placa, foi utilizado um cabo Ethernet (Figura 1 – G) conectado entre a porta Ethernet da placa e a porta Ethernet do computador. Em seguida, o firmware desenvolvido foi

passado para a BBB na Máquina Virtual Linux por meio do software FileZila e assim, executado para testes.

3.3 Desenvolvimento e testes de um Firmware escravo Modbus TCP e mestre Modbus RTU.

Desenvolveu-se um Firmware capaz de fazer requisições para equipamentos escravos da rede, no protocolo Modbus RTU e receber comandos de um mestre no protocolo Modbus TCP, configurando assim, o *Gateway* Modbus TCP/RTU. A partir deste momento, substituiu-se a linguagem de programação C pelo Python. Consequentemente, os firmwares foram baseados em outra biblioteca. A validação do funcionamento do Firmware foi feita através do software simulador escravo Modbus RTU ModSim e mestre Modbus ModScan. No teste, para estabelecer a conexão RS-232/485 entre o simulador ModSim e a BBB, conectaram-se os pinos da, já previamente configurada, UART 4 da BBB aos pinos correspondentes da placa FTDI (Figura 1 – A), enquanto a porta Micro USB da placa FTDI foi conectada à porta USB do computador. Para estabelecer a conexão Modbus TCP entre BBB e simulador ModScan, conectou-se o cabo Ethernet entre a entrada Ethernet da BBB e do computador. Em seguida, o firmware desenvolvido foi passado para a BBB na Máquina Virtual Linux por meio do software FileZila e assim, executado para testes.

3.4 Testes do Firmware mestre Modbus RTU e escravo Modbus TCP fazendo uso de adaptadores RS485.

Montou-se um experimento para a comunicação do firmware mestre Modbus RTU e escravo Modbus TCP com os mesmos simuladores, porem desta vez, fazendo uso de adaptadores RS485. O experimento apresentou a mesma montagem da etapa anterior, com a diferença que para estabelecer a conexão RTU, conectou-se a UART4 com um circuito que adaptaria RS485/RS232, a fim de verificar se o circuito responsável pela conversão da porta serial da BBB para o barramento RS485 funcionaria, já que essa conexão será exigida quando o programa funcionar em seu estágio final. Para isso, foi utilizado o modulo de conversão RS485 (Figura 1 – D) conectado à porta UART4. Como o modulo RS485 utiliza o padrão de 5v e a BBB, 3v3, foi necessário, para converter 3v3 em 5v, o conversor de nível lógico (Figura 1 – B). A saída do modulo,

foi adaptada para RS232 com o modulo conversor RS232 - RS485 (Figura 1 -C) e daí conectada ao computador via USB usando o adaptador USB/Serial Conversor RS232.

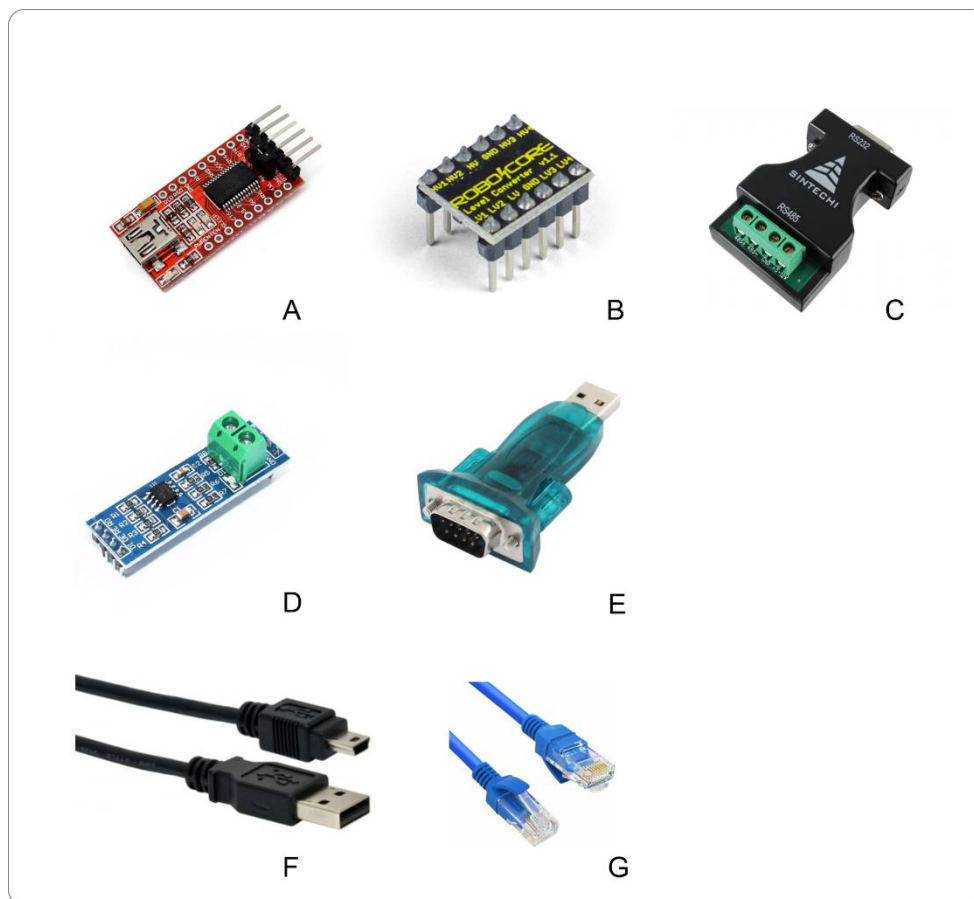


Figura 1 – Componentes utilizados e citados na seção de metodologia do trabalho. (A) Conversor USB Serial FT232RL FTDI; (B) Conversor de Nível Lógico RC; (C) Conversor RS232 - RS485; (D) Conversor de dados TTL para RS485; (E) Adaptador USB 2.0 Serial Conversor RS232 DB9; (F) Cabo Mini USB; (G) Cabo Ethernet

4 RESULTADOS E DISCUSSÃO

Todos os itens citados na seção 3 foram realizados, sendo seus respectivos resultados apresentados abaixo:

4.1 Desenvolvimento de um Firmware básico na Beaglebone Black

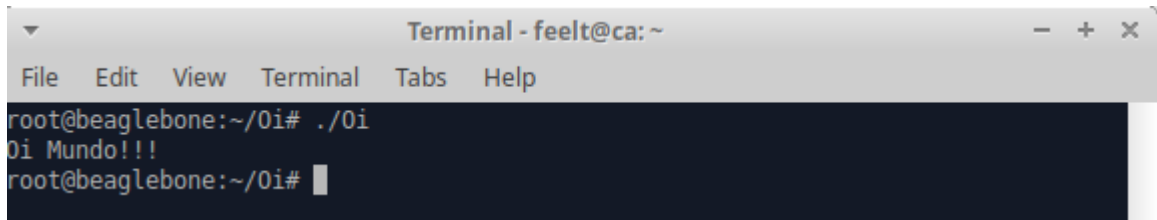
O desenvolvimento de um firmware básico utilizando a BBB foi o primeiro objetivo prático a ser alcançado. Após leitura dos manuais de estudo elaborados pelo grupo de pesquisa e colaboradores do LASEC, foi desenvolvido um programa básico que imprime na tela a mensagem “Oi mundo!!!”. Esse momento foi importante para se familiarizar com o ambiente da Máquina Virtual Linux e as particularidades do sistema operacional embarcado na BBB. As telas do programa estão representadas nas figuras 1 e 2.

A screenshot of a code editor window titled "/home/feelt/Desktop/Apostilas BeagleBone/01 - Oi Mundo!!! (Hello World!!!)/BeagleBone Black/Oi.c - Mousepad". The editor contains the following C code:

```
#include <stdio.h>

int main()
{
    printf("Oi Mundo!!! \n");
}
```

Figura 2 – Código em C que imprime a mensagem “Oi mundo!!!” na tela.

A screenshot of a terminal window titled "Terminal - felt@ca: ~". The terminal shows the execution of the program. The prompt is "root@beaglebone:~/Oi#". The user enters "./Oi", and the output is "Oi Mundo!!!". The prompt returns to "root@beaglebone:~/Oi#".

```
root@beaglebone:~/Oi# ./Oi
Oi Mundo!!!
root@beaglebone:~/Oi#
```

Figura 3 – Tela resultante do programa que imprime na tela a mensagem “Oi mundo!!!”.

4.2 Desenvolvimento e testes de um Firmware escravo Modbus TCP

O desenvolvimento do primeiro Firmware voltado ao protocolo Modbus foi um programa que atua na rede como escravo TCP. Várias bibliotecas open-source disponíveis online suportam a comunicação Modbus TCP como escravo. Por essa razão, foi o ponto de partida do desenvolvimento do Gateway. A biblioteca escolhida como referência foi a FreeModbus, e foi nela baseada a construção do firmware escravo Modbus TCP. Após compreensão do código e edição da biblioteca, foram implementados testes com os simuladores ModsScan e Qmodbus que por sua vez, apresentaram resultados positivos quanto a comunicação e estabelecimento da rede entre simulador e BBB. As telas de simulação e programação estão representadas nas figuras 4 a 7.


```

/home/feelt/Documents/GatewayModbus/ModBusGateway-master/demo.c - Mousepad
File Edit Search View Document Help

static void* pvPollingInread(void* pvParameter);
static BOOL bInitializeGPIO();
static int initrff();
USHORT getstatusportmodbus(void);
USHORT getMBAddress(void);

/* ----- Implementação das funções para requerimento do Modbus ----- */
//static BOOL bWriteGPIO(USHORT value1, USHORT value2);
//static int bWriteAD(USHORT value1);

/* ----- Start implementation ----- */
static int initrff(void){
    usRegHoldingBuf[0] = 110;
    usRegHoldingBuf[1] = 220;
    usRegHoldingBuf[2] = 330;
    usRegHoldingBuf[3] = 440;
    return 0;
}

int
main( int argc, char *argv[] )
{
    int          iExitCode;
    CHAR          cCh;
    BOOL          bDoExit;
    USHORT        stPortMod;
    int           count;
    USHORT        mbaddr;

```

Figura 4 – Trecho do firmware escravo Modbus TCP.

```

root@beaglebone:~/freemodbus-master/LINUXTCP# ./tcpmodbus
/sys/class/gpio/gpio50/value
> h
FreeModbus demo application help:
'd' ... disable protocol stack.
'e' ... enabled the protocol stack
's' ... show current status
'q' ... quit applicationr
'h' ... this information

Copyright 2007 Steven Guo <gotop167@163.com>
> s
Protocol stack is running. PortModbus=ETH0 MyMBAddress=255
>

```

Figura 5 - Tela resultante do firmware escravo Modbus TCP.

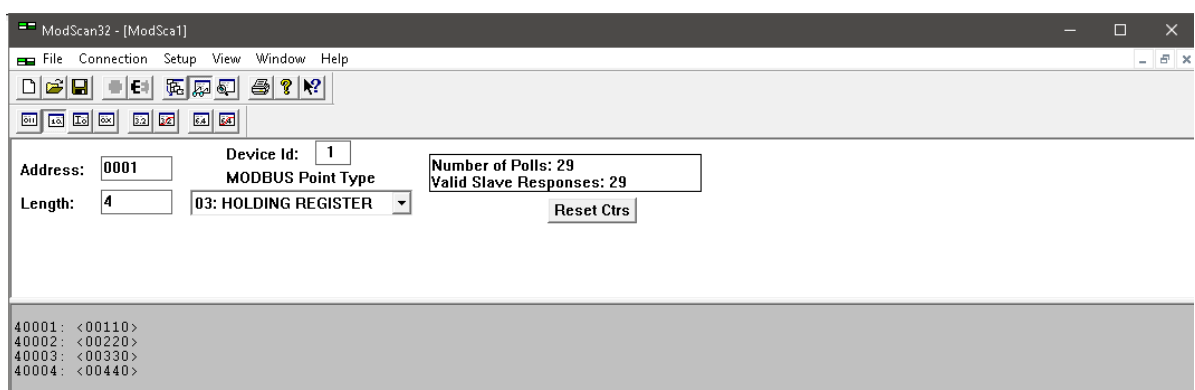


Figura 6 – Tela do programa simulador ModScan.

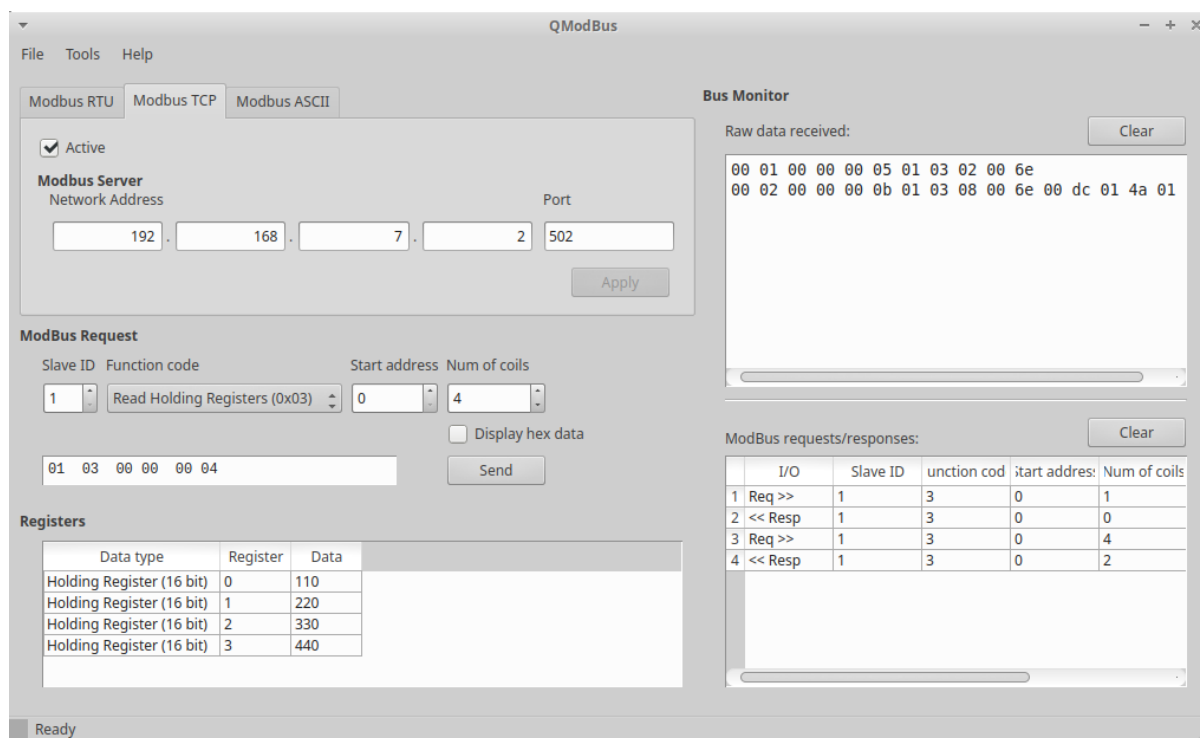


Figura 7 – Tela do programa simulador QModbus.

4.5 Desenvolvimento e testes de um Firmware escravo Modbus TCP e mestre Modbus RTU.

Este momento da pesquisa foi marcado pela mudança na linguagem de programação. Essa mudança aconteceu após perceber que grande maioria das bibliotecas open-source na linguagem C não tinham suporte para operar como mestre Modbus, apenas escravo. Foram feitas tentativas de adaptação do código, mas, devido à complexidade, nenhuma das tentativas teve êxito. Surge então a alternativa de desenvolver o firmware em Python. Encontrou-se na plataforma GitHub, uma biblioteca em Python que se mostrou mais simples, mais concisa e até mais funcional que a bibliotecas antes utilizada. Nesse sentido, começou-se a desenvolver o firmware Gateway Modbus baseado nessa biblioteca, que por sua vez já integrava a função Mestre Modbus RTU e Escravo Modbus TCP no mesmo firmware. Dessa forma, poucas adaptações foram feitas para que o programa funcionasse com nosso sistema de simuladores. As telas desta etapa da pesquisa estão nas figuras 8 e 9.

```

/home/feelt/Documents/GatewayModbus/ModBusGateway-master/modbus-gateway.py - Mousepad
File Edit Search View Document Help
61 # send the ModbusRTU request
62 self.serial.write(rtu_request)
63 # read first three bytes of the response to check for errors
64 rtu_response = self.serial.read(3)
65 if ord(rtu_response[1]) > 0x80:
66     logger.debug("RTU Error Response {}".format(":".join("{:02X}".format(ord(c)) for c in rtu_response)))
67     tcp_response = tcp_request[0:5] + chr(3) + rtu_response
68     logger.debug("TCP Error Response {}".format(":".join("{:02X}".format(ord(c)) for c in tcp_response)))
69     self.request.sendall(tcp_response)
70 else:
71     # if no error, read number of bytes indicated in RTU response
72     rtu_response += self.serial.read(ord(rtu_response[2]) + 2)
73     logger.debug("RTU Response {}".format(":".join("{:02X}".format(ord(c)) for c in rtu_response)))
74     # convert ModbusRTU response into a Modbus TCP response
75     tcp_response = tcp_request[0:5] + chr(ord(rtu_response[2]) + 2) + rtu_response[2:]
76     logger.debug("TCP Response {}".format(":".join("{:02X}".format(ord(c)) for c in tcp_response)))
77     # return converted TCP response
78     self.request.sendall(tcp_response)
79
80 def finish(self):
81     self.serial.close()
82
83 if __name__ == "__main__":
84     config = ConfigParser.RawConfigParser()
85     config.read('modbus-gateway.cfg')
86     address = (config.get("ModbusTCP", "host"), config.getint("ModbusTCP", "port"))
87     server = SocketServer.TCPServer(address, ModbusGateway)
88     server.serve_forever()
89
90

```

Figura 8 – Trecho do código .py mestre Modbus TCP e escravo Modbus RTU.

```

/home/feelt/Documents/GatewayModbus/ModBusGateway-master/modbus-gateway.cfg - Mousepad
File Edit Search View Document Help
1 [ModbusTCP]
2 host=192.168.0.50
3 port=502
4
5 [ModbusRTU]
6 port=/dev/tty04
7 baudrate=9600
8 bytesize=8
9 stopbits=1
10 parity=N
11 timeout=5
12 gpio=7
13

```

Figura 9 - Trecho do código .cfg mestre Modbus TCP e escravo Modbus RTU

No teste com os simuladores computacionais ModScan e ModSim, o firmware teve um resultado bastante positivo, uma vez que a conexão tanto no protocolo RTU como TCP foi obtida com sucesso, como mostram as figuras 10 a 12.

```

Terminal - felt@ca: ~
File Edit View Terminal Tabs Help

:02
2014-05-14 23:29:30,508: DEBUG - RTU Request 01:03:00:00:00:02:C4:0B
2014-05-14 23:29:30,593: DEBUG - RTU Response 01:03:04:00:30:00:0F:BA:38
2014-05-14 23:29:30,597: DEBUG - TCP Response 52:00:00:00:00:06:01:03:04:00:3
0:00:0F
2014-05-14 23:29:31,504: DEBUG - TCP Request 53:00:00:00:00:06:01:03:00:00:00
:02
2014-05-14 23:29:31,514: DEBUG - RTU Request 01:03:00:00:00:02:C4:0B
2014-05-14 23:29:31,608: DEBUG - RTU Response 01:03:04:00:30:00:0F:BA:38
2014-05-14 23:29:31,612: DEBUG - TCP Response 53:00:00:00:00:06:01:03:04:00:3
0:00:0F
2014-05-14 23:29:32,508: DEBUG - TCP Request 54:00:00:00:00:06:01:03:00:00:00
:02
2014-05-14 23:29:32,516: DEBUG - RTU Request 01:03:00:00:00:02:C4:0B
2014-05-14 23:29:32,610: DEBUG - RTU Response 01:03:04:00:30:00:0F:BA:38
2014-05-14 23:29:32,617: DEBUG - TCP Response 54:00:00:00:00:06:01:03:04:00:3
0:00:0F
2014-05-14 23:29:33,509: DEBUG - TCP Request 55:00:00:00:00:06:01:03:00:00:00
:02
2014-05-14 23:29:33,514: DEBUG - RTU Request 01:03:00:00:00:02:C4:0B
2014-05-14 23:29:33,608: DEBUG - RTU Response 01:03:04:00:30:00:0F:BA:38
2014-05-14 23:29:33,612: DEBUG - TCP Response 55:00:00:00:00:06:01:03:04:00:3
0:00:0F

```

Figura 10 - Tela resultante do firmware escravo Modbus TCP e mestre Modbus RTU.

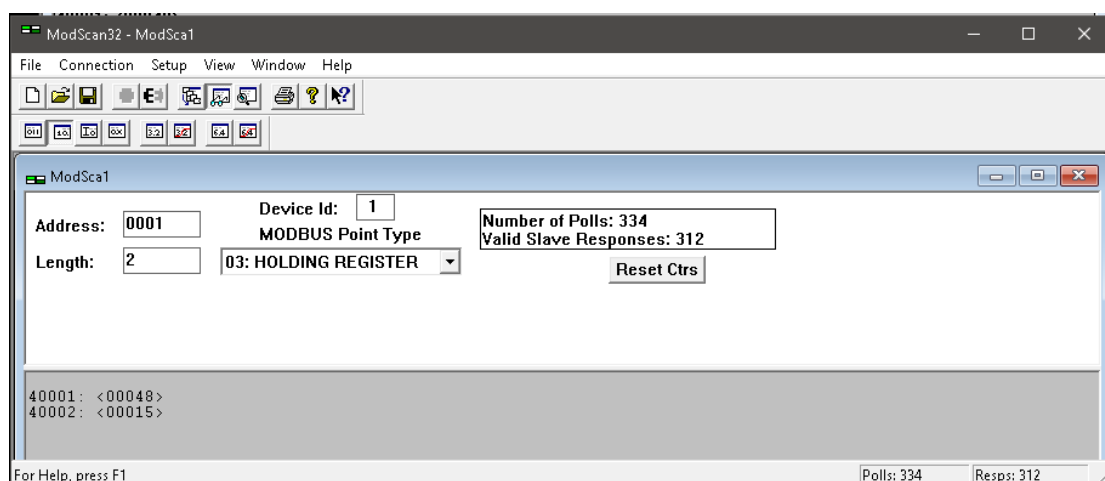


Figura 11 - Tela do programa simulador ModScan.

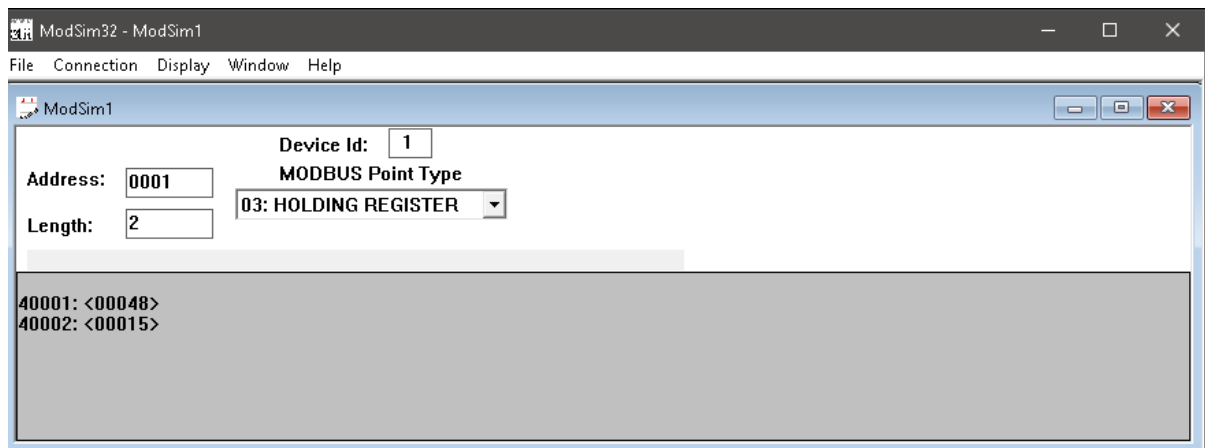


Figura 12 – Tela do programa simulador ModSim.

4.6 Testes do Firmware mestre Modbus RTU e escravo Modbus TCP fazendo uso de adaptadores RS485.

Após o resultado positivo dos testes do firmware Gateway Modbus no ambiente de simulação computacional, esperava-se que a integração do circuito como um todo, com os adaptadores RS485 tivessem o mesmo desfecho dos testes. Entretanto, por mais que o firmware já tenha se mostrado funcional, o circuito que adapta a porta serial UART4 da BBB para um barramento RS485 não funcionou corretamente uma vez que, os simuladores e o próprio sistema de Debug do firmware apontaram problemas na conexão entre mestre e escravo.

5 PERSPECTIVAS FUTURAS

Com os resultados apresentados, o foco do trabalho está em solucionar o problema no circuito adaptador RS485, já que essa conexão é imprescindível para operação final do Gateway proposto. Novos componentes e novas táticas de adaptação serão implementadas até que o circuito funcione como o esperado. Quando esse ponto for superado, o funcionamento do Gateway será colocado a prova com a rede de equipamentos do laboratório, para a partir daí analisar objetivamente sua eficiência e aplicabilidade num contexto industrial real. As próximas atividades seguirão o cronograma abaixo:

Fase Tarefa

- 5.1 Solução do problema no circuito de adaptação RS485.
 - 5.2 Integração do Firmware com a rede de equipamentos do laboratório.
 - 5.3 Testes finais do sistema.
 - 5.4 Documentação do código e preparação do relatório final.
-

	Jun	Jul	Ago	Set
5.1	X			
5.2		X		
5.3			X	X
5.4				X

Tabela 1 – Cronograma de trabalho para conclusão do projeto.

6 REFERÊNCIAS

- [1] Drivers Control. Industrial Ethernet powers ahead, while fieldbuses slide. Drivers Control. 2020. Jun.[acesso em 6 out 2021] Disponível em: <https://drivesncontrols.com/news/fullstory.php/aid/6396/IndustrialEthernetpowersaheadwhilefieldbusesslide>
- [2] Tabaa M, Chouri B, Saadaoui S, Alami K. Industrial Communication based on Modbus and Node-Red. Procedia Computes Science; Vol. 130. 2018. p. 583 – 588
- [3] Constantino, S. Protocolos Orientados a Caracter. Belo Horizonte: Departamento de Engenharia Eletrônica, 2002. p. 2-4.
- [4] Grygoruk A, Legierski J. IoT gateway – Implementation proposal based on Arduino Board. Anais de Computer Science and Information Systems, 2016-10-01, Vol.8, p.1011-1014.
- [5] Sravani, B. The Critical Role of Gateways in IoT. Mouser Electronics. 2019. Nov. [acesso em 6 out 2021]; Disponível em: <https://br.mouser.com/blog/blog/blog/critical-role-of-gateways-iot>