Universidade Federal de Campina Grande - UFCG

Centro de Engenharia Elétrica e Informática - CEEI

Departamento de Engenharia Elétrica - DEE

Relatório de Estágio Supervisionado
Laboratório EMBEDDED

Andresso da Silva

Campina Grande, PB

21 de Maio de 2021

Andresso da Silva

Relatório de Estágio Supervisionado
Laboratório EMBEDDED

Relatório de Estágio Supervisionado submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Bacharel em Engenharia Elétrica.

Orientador: Jaidilson Jó da Silva, D.Sc.

Supervisor: George Acioli Júnior, D.Sc.

Campina Grande, PB

21 de Maio de 2021

Andresso da Silva

Relatório de Estágio Supervisionado
Laboratório EMBEDDED

Relatório de Estágio Supervisionado submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Bacharel em Engenharia Elétrica.

Trabalho aprovado em: ____ / ____ / ____

Jaidilson Jó da Silva, D.Sc.
Orientador

Danilo Freire de Souza Santos
Professor Convidado

Campina Grande, PB

21 de Maio de 2021

Lista de ilustrações

Figura 1 – Fotografia da Fachada do Laboratório Embedded 3
Figura 2 – Representação da pilha de comunicação do *Modbus*. 6
Figura 3 – Representação do *frame Modbus* geral. 6
Figura 4 – Representação da troca de mensagens entre cliente e servidor quando não há erro. 7
Figura 5 – Representação da troca de mensagens entre cliente e servidor quando há erro. 7
Figura 6 – Representação do *frame Modbus* TCP. 7
Figura 7 – Representação das mensagens trocadas entre cliente e servidor durante o envio do comando para definir a tangente de phi como 0,75. 22
Figura 8 – Representação das mensagens trocadas entre cliente e servidor durante o envio do comando para definir o comando para tangente de phi. 22

Lista de tabelas

Tabela 1 – Funções de leitura e escrita em registradores do protocolo *Modbus*. 8
Tabela 2 – Endereços das tabelas de armazenamento de um servidor *Modbus*. 9
Tabela 3 – Funções de leitura e escrita em registradores da biblioteca *PyModbus*. . . . 10
Tabela 4 – Parâmetros possíveis de acessar via visor do inversor. 12
Tabela 5 – Alguns dos parâmetros do inversor. 13
Tabela 6 – Endereços dos registradores responsáveis por receber o comando e os dados. 13
Tabela 7 – Lista de comandos. 14
Tabela 8 – Frame de dados para definir a tan(ϕ) como 0.75 utilizando a função 0x06. . 14
Tabela 9 – Frame de dados para definir a tan(ϕ) como 0.75 utilizando a função 0x10. . 15
Tabela 10 – Frame de dados para parar o inversor. 15
Tabela 11 – Frame de dados para iniciar o inversor. 15
Tabela 12 – Frame de dados para definir modo standby do inversor. 16
Tabela 13 – Frame de dados para definir a cos(ϕ) como -0.8. 16
Tabela 14 – Comandos para definir parâmetros da bateria no inversor. 17
Tabela 15 – Frame de dados para definir a corrente da bateria como 20 A. 17
Tabela 16 – Frame de dados para definir a tangente de phi como 0,75 utilizando a função 0x10. 21
Tabela 17 – Frame de dados para iniciar o inversor. 23
Tabela 18 – Frame de dados para parar o inversor. 23

Lista de abreviaturas e siglas

CEEI	Centro de Engenharia Elétrica e Informática
EMDEDDED	Laboratório de Sistemas Embarcados e Computação Pervasiva
P&D	Pesquisa e Desenvolvimento
UFCG	Universidade Federal de Campina
ADU	Application Data Unit
PDU	Protocol Data Unit
HDLC	High Leve Data Link Control
IP	Internet Protocol
MD	MODBUS Protocol
MBAP	MODBUS Application Protocol

Sumário

1 INTRODUÇÃO 2

2 APRESENTAÇÃO DO AMBIENTE DE TRABALHO 3
2.1 O Embedded 3
2.2 Projeto de Pesquisa e Desenvolvimento 3

3 FUNDAMENTAÇÃO TEÓRICA 5
3.1 Inversores de Frequência 5
3.2 Protocolo *Modbus* 5
3.3 Biblioteca *PyModbus* 8
3.3.1 Servidor 8
3.3.2 Cliente 9

4 ATIVIDADE DESENVOLVIDAS 11
4.1 Investigação dos Parâmetros do Inversor 11
4.2 Investigação do Comandos do Inversor 12
4.3 Exemplos de Comando para o Inversor 14
4.3.1 Tangente de *Phi* 14
4.3.2 Parar Inversor 15
4.3.3 Iniciar Inversor 15
4.3.4 Inversor em *Standby* 16
4.3.5 Cosseno de *Phi* 16
4.3.6 Parâmetros da Bateria 16
4.4 Desenvolvimento do Protótipo de Servidor *Modbus* 17
4.5 Desenvolvimento do Protótipo de Cliente *Modbus* 18
4.6 Testes com o Protótipo do Cliente *Modbus* 21

5 CONSIDERAÇÕES FINAIS 24

REFERÊNCIAS 25

1 Introdução

O Estágio Integrado ou Supervisionado é obrigatório para a conclusão do curso de Engenharia Elétrica. Isso se deve à necessidade de complementar a formação do aluno com experiência prática na área profissional, de forma que conceitos aprendidos durante o curso sejam aplicados.

Nesse contexto, neste trabalho estão descritas as principais atividades desenvolvidas pelo estudante de graduação em Engenharia Elétrica, Andresso da Silva, durante o estágio supervisionado no Laboratório de Sistemas Embarcados e Computação Pervasiva (Embedded) da UFCG. O estágio foi realizado no período entre 08 de Março a 21 de Maio de 2021, com uma carga horária de 20 horas semanais, totalizando 214 horas, sob a orientação do professor Jaidilson Jó da Silva e a supervisão do professor George Acioli Júnior.

O estágio foi realizado em um projeto de Pesquisa e Desenvolvimento (P&D) do laboratório e as atividades desenvolvidas foram relacionadas à investigação e desenvolvimento de *software* para a comunicação com um inversor de frequência utilizando o protocolo *Modbus*. No estágio supervisionado, o estagiário desenvolveu um protótipo de cliente *Modbus* em *Python* utilizando a biblioteca *PyModbus* a fim de ler e definir parâmetros do inversor.

Os demais capítulos desse trabalho estão organizados da seguinte forma: no capítulo 2 encontra-se a apresentação do laboratório e do projeto em que o estágio foi realizado; no capítulo 3 são apresentados conceitos fundamentais para a realização das atividades; no capítulo 4 são descritas as atividades desenvolvidas pelo aluno; e no capítulo 5 são apresentadas as considerações finais do trabalho.

2 Apresentação do Ambiente de Trabalho

Neste capítulo serão apresentados o laboratório Embedded e a estrutura organizacional do projeto de pesquisa e desenvolvimento em que o estágio foi realizado.

2.1 O Embedded

O Laboratório de Sistemas Embarcados e Computação Pervasiva (Embedded) é um dos laboratórios do Centro de Engenharia Elétrica e Informática (CEEI) da Universidade Federal de Campina Grande (UFCG), em Campina Grande, Paraíba, Brasil. Na Figura 1 pode ser vista a fachada do laboratório. A equipe do laboratório é formada por professores e pesquisadores do CEEI que desenvolvem projetos com alunos de doutorado, mestrado e graduação [1].

Figura 1 – Fotografia da Fachada do Laboratório Embedded

Fonte: embedded.ufcg.edu.br

2.2 Projeto de Pesquisa e Desenvolvimento

O estagiário realizou as atividades em um projeto de cooperação técnico e científico entre o laboratório e uma empresa parceira. No projeto foram executadas atividades de pesquisa e

desenvolvimento de *software*, incluindo investigação acerca de um inversor de frequência e implementação de soluções para comunicação via *Modbus*.

Nesse capítulo foram apresentados o laboratório em que foi realizado o estágio e o projeto em que o aluno estava inserido. No próximo capítulo são apresentados os conceitos e definições essenciais para o desenvolvimento das atividades realizadas durante o estágio.

3 Fundamentação Teórica

Neste capítulo serão apresentados os principais conceitos, definições e características dos inversores de frequência, do protocolo Modbus e da biblioteca *PyModbus* necessários para o desenvolvimento das atividades do estágio.

3.1 Inversores de Frequência

O inversor de frequência é um dispositivo de eletrônica de potência utilizado na conversão de corrente contínua (CC) para a corrente alternada (CA). Os inversores de frequência são utilizados em sistemas fotovoltaicos para converter de forma eficiente a energia gerada pelos painéis solares em energia adequadas para o uso nas redes residenciais ou industriais. A utilização de inversores garante maior segurança para o sistema e permite o monitoramento da eletricidade gerada [2].

Existem basicamente dois tipos de inversores utilizados na conversão de energia solar residencial, o inversor *string* e o micro-inversor.

O inversor *string* é o mais comum e mais barato de ser utilizado. Isso se deve ao fato de que os painéis solares são agrupados e conectados a um inversor que faz o controle de forma centralizada. Como não há o controle individual dos painéis, em algumas situações não é o sistema mais eficiente de conversão, como quando há sombras em determinados painéis.

O micro-inversor é utilizado em cada uma célula fotovoltaica, com o objetivo de estabilizar a energia desses elementos para que funcionem independentes dos locais, aumentando a eficiência. Esse tipo de inversor é mais caro e é mais indicado para os sistemas que precisam otimizar de forma individual a energia gerada em cada célula.

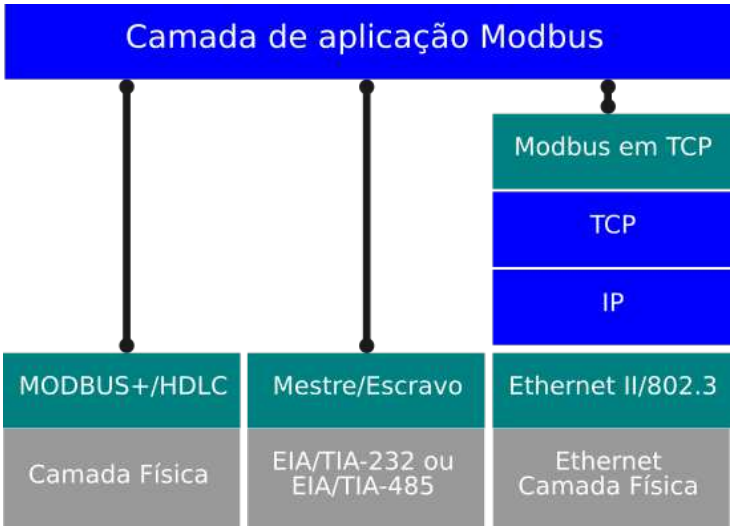
Inversores são cada vez mais importantes com o crescimento da utilização de energias alternativas como a energia solar. O inversor pode auxiliar no controle da energia utilizada, principalmente para aqueles que fazem um uso limitado dessa fonte. Além da ajudar na redução do aquecimento global, seu custo de manutenção não é alto e pode ter uma eficiência de até 98%.

3.2 Protocolo *Modbus*

O *Modbus* é um protocolo de camada de aplicação para a troca de mensagens entre clientes e servidores que podem ser dispositivos de diferentes naturezas além de estarem conectados

em diferentes redes. Ele pode ser implementado por meio de TCP/IP, por meio de transmissão serial assíncrona ou por meio do *Modbus* Plus, como apresentado na Figura 2.

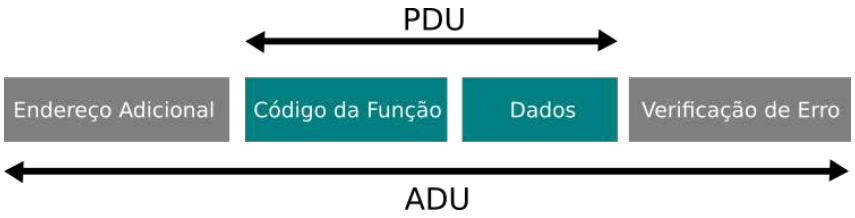
Figura 2 – Representação da pilha de comunicação do *Modbus*.



Fonte: Adaptado de [3]

O *Modbus* é um protocolo padrão para a indústria desde os anos 1979 e funciona por meio de um modelo de requisição e resposta que utilizam códigos associados às funções. As mensagens trocadas constituem um protocolo de unidade de dados (PDU) que é independente das camadas de comunicação dos dispositivos e é constituído pelo código da função e os dados. Esse PDU com mais campos como os endereços necessários e a verificação de erros formam o unidade de dados de aplicação (ADU) que é apresentado no diagrama da Figura 3.

Figura 3 – Representação do *frame Modbus* geral.

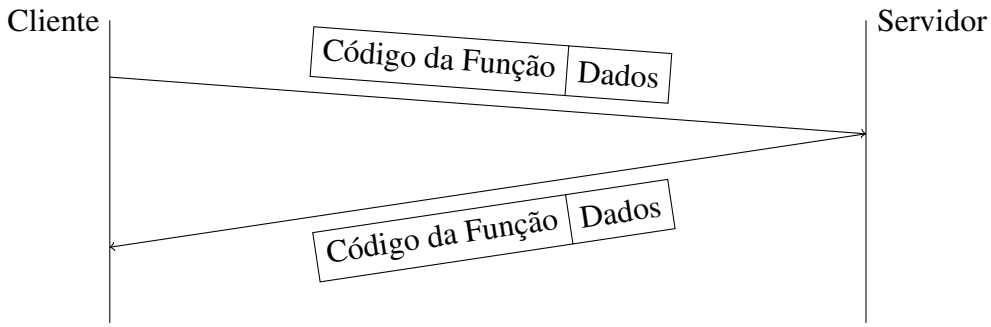


Fonte: Adaptado de [3].

O cliente produz um ADU como o apresentado no diagrama da Figura 3 e envia para o servidor para iniciar a conexão via *Modbus*. A ação desejada é definida por meio do *Código da Função* que pode assumir um valor entre 1 e 255 em decimal, ocupando um byte. Nesse intervalo, a função 0 não é válida e as funções com os códigos entre 128 e 255 são reservadas para exceções que podem ocorrer. O campo *Dados* é usado para complementar o código da função e informar ao servidor qual a ação desejada (*e.g.*, endereços de registradores, quantidade de itens enviados e a quantidade de bytes desses itens).

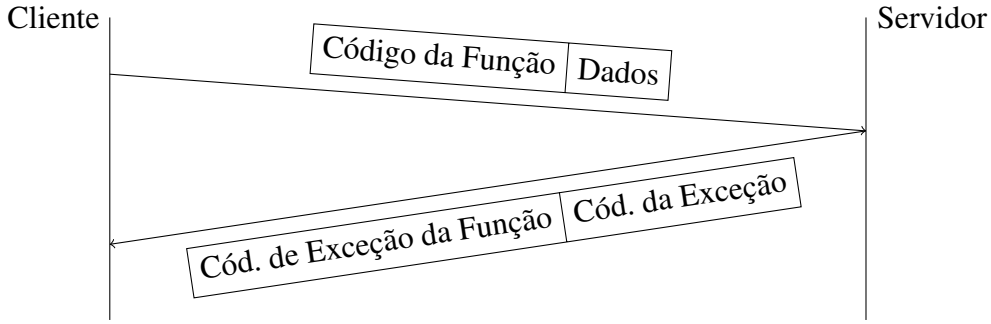
O fluxo de mensagens trocadas entre o cliente e o servidor quando não há erros é apresentado no diagrama da Figura 4. O cliente envia o código da função e os dados e o servidor responde com o mesmo código da função e os dados.

Figura 4 – Representação da troca de mensagens entre cliente e servidor quando não há erro.



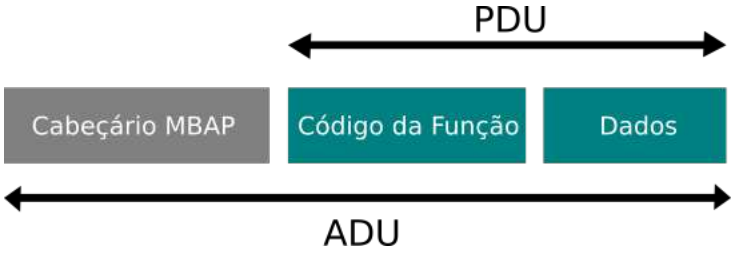
O fluxo de mensagens trocadas entre o cliente e o servidor quando há erros é diferente de quando não há e é apresentado no diagrama da Figura 5. O cliente envia o código da função e os dados e o servidor responde com o código de exceção associado à função e o código da exceção.

Figura 5 – Representação da troca de mensagens entre cliente e servidor quando há erro.



Quando se trata do *Modbus* TCP, os *frames* trocados entre o cliente e o servidor tomam a forma de como é apresentado no diagrama da Figura 6.

Figura 6 – Representação do *frame Modbus* TCP.



Fonte: Adaptado de [4].

O cabeçário MBAP (*Modbus Application Protocol*) é utilizado para identificar o ADU.

Para ler e escrever nos registradores, é possível utilizar algumas das funções públicas do protocolo *Modbus* apresentadas na Tabela 1.

Tabela 1 – Funções de leitura e escrita em registradores do protocolo *Modbus*.

Função	**Código da Função**
Ler registradores	0x03
Escrever em registrador	0x06
Escrever em registradores	0x10

É possível ler o conteúdo de registradores utilizando a função 0x03, escrever em um registrador utilizando a função 0x06 ou escrever em múltiplos registradores utilizando a função 0x10.

3.3 Biblioteca *PyModbus*

Para implementar a comunicação com o inversor, foi escolhida a biblioteca *PyModbus*[1] versão 2.5.0 que implementa de forma completa o protocolo *Modbus* na linguagem *Python* (3.6). Essa biblioteca funciona para qualquer versão do *Python* acima da versão 2.7, foi a implementação mais completa encontrada e a melhor documentada. Por meio de outras bibliotecas da linguagem como *twisted*[2], *torndo*[3] e *asyncio*[4] é possível fazer aplicações de comunicação assíncrona [5].

Para instalar o *PyModbus* com suporte do twisted basta utilizar o comando *pip install -U pymodbus[twisted]*.

Devido ser implementada em *Python*, é possível fazer protótipos rapidamente, realizar testes e integrar com outros sistemas, além de permitir consultar diferentes dispositivos na rede para obter status via *Modbus*.

3.3.1 Servidor

O servidor *PyModbus* pode funcionar como um servidor *Modbus* completo e permite que um usuário teste os diversos dispositivos conectados a ele, sendo o número de dispositivos possíveis limitado somente pelo número de IP disponíveis. O *PyModbus* dá suporte a servidores do tipo TCP, UDP, ASCII Serial, RTU Serial e Binário Serial que podem assíncronos ou síncronos, permitindo acesso às informações dos dispositivos conectados.

[1] Disponível em: https://pymodbus.readthedocs.io/en/latest/
[2] Disponível em: https://twistedmatrix.com/trac/
[3] Disponível em: https://www.tornadoweb.org/en/stable/
[4] Disponível em: https://docs.python.org/3/library/asyncio.html

Um servidor *Modbus* tem um conjunto de registradores chamados de tabelas apresentados na Tabela 2. Para implementar um servidor utilizando o *PyModbus* de forma que esse servidor possua essas tabelas, é necessário definir alguns parâmetros como no Algoritmo 3.1.

Tabela 2 – Endereços das tabelas de armazenamento de um servidor *Modbus*.

Número do Registrador	**Endereço do registrador em Hexadecimal**	**Tipo**	**Nome**	**ID**
1-9999	0000 a 270E	Leitura/Escrita	Bobinas de Saída Discreta	DO
10001-19999	0000 a 270E	Leitura	Contatos de Entrada Discretos	DI
30001-39999	0000 a 270E	Leitura	Registradores Analógicos de Entrada	AI
40001-49999	0000 a 270E	Leitura/Escrita	Registradores de retensão de saída	AO

As tabelas são definidas por meio da função *ModbusSequentialDataBlock()* que recebe como parâmetro o endereço inicial de memória e o número de registradores por meio de uma lista com o número de elementos desejados, como no Algoritmo 3.1.

```
store = ModbusSlaveContext(
    di = ModbusSequentialDataBlock(0, [0]*2000),
    co = ModbusSequentialDataBlock(0, [0]*2000),
    hr = ModbusSequentialDataBlock(0, [0]*2000),
    ir = ModbusSequentialDataBlock(0, [0]*2000)
)
# Define contexto
context = ModbusServerContext(slaves=store, single=True)
```

Algoritmo 3.1 – Definição das tabelas Servidor utilizando o *PyModbus*.

Com o contexto definido, é possível ter acesso aos registradores do servidor.

3.3.2 Cliente

O cliente *PyModbus* possui também uma implementação completa que permite ler e escrever nos registradores dos servidores *Modbus*, além de possuir outras funcionalidades como diagnósticos. Como nos servidores, o *PyModbus* dá suporte a clientes do tipo TCP, UDP, Serial ASCII, Serial RTU e Serial Binary, que podem ser síncronos ou assíncronos.

Para iniciar uma sessão utilizando um cliente feito utilizando o *PyModbus* é necessário informar o IP do servidor *Modbus* e a porta, como apresentado no Algoritmo 3.2.

```
client = ModbusClient('localhost', port=5020)
client.connect()
```

Algoritmo 3.2 – Início de uma Sessão *Modbus* utilizando um cliente *PyModbus*.

No instanciamento do cliente, a biblioteca ainda permite a especificação do método (RTU, Binário, ASCII) e outros parâmetros relacionados à conexão. Os métodos de leitura e escrita em registradores implementadas no *PyModbus* são métodos do objeto client e são apresentadas na Tabela 3.

Tabela 3 – Funções de leitura e escrita em registradores da biblioteca *PyModbus*.

Função	**Código da Função**	**Função *PyModbus***
Ler registradores	0x03	read_holding_registers()
Escrever em registrador	0x06	write_register()
Escrever em registradores	0x10	write_registers()

Desta forma, para escrever o valor 10 no registrador 1 e posteriormente ler o valor do mesmo endereço, o código seria como o apresentado no Algoritmo 3.3. O parâmetro UNIT está relacionado ao endereço adicional necessário para indicar de qual cliente o comando está vindo.

```
1 rq = client.write_register(1, 10, unit=UNIT)
2 rr = client.read_holding_registers(1, 1, unit=UNIT)
```

Algoritmo 3.3 – Escrevendo no registrador 1 o valor 10 e lendo o valor.

Para escrever o valor 10 em múltiplos registradores, o código seria como o apresentado no Algoritmo 3.3. Nesse caso, é necessário informar o endereço do primeiro registrador e uma lista contendo os valores que se deseja armazenar nos registradores. Os endereços são contíguos, então será armazenado o valor 10 nos 8 registradores começando com o de endereço 1. Para ler os valores armazenados nos registradores, é necessário informar o endereço do primeiro registrador e a quantidade de registradores que se deseja ler.

```
1 rq = client.write_registers(1, [10]*8, unit=UNIT)
2 rr = client.read_holding_registers(1, 8, unit=UNIT)
```

Algoritmo 3.4 – Escrevendo o valor 10 em 8 registradores a partir do 1 e lendo os valores.

Para verificar se houve erro durante o envio ou na execução dos comandos, pode-se utilizar a função *isError()*, como apresentado no Algoritmo 3.5, além de verificar se o conteúdo lido é igual ao conteúdo que se desejava escrever nos registradores.

```
1 assert(not rq.isError())
2 assert(not rr.isError())
3 assert(rr.registers == [10]*8)
```

Algoritmo 3.5 – Verificando se houve erro na execução dos comandos.

Nesse capítulo foram apresentados os conceitos essenciais para a realização das atividades durante o estágio envolvendo inversores de frequência, *Modbus* e a biblioteca *PyModbus*. No próximo capítulo são descritas as atividades desenvolvidas pelo aluno durante o estágio.

4 Atividade Desenvolvidas

Neste capítulo serão apresentadas as principais atividades desenvolvidas durante o estágio. As atividades envolveram a investigação de quais os parâmetros passíveis de leitura e modificação via protocolo *Modbus*, os comandos necessários para ler e modificar os parâmetros, o desenvolvimento do protótipo do cliente *Modbus* para se comunicar com o inversor e os testes da comunicação.

Foi utilizada a metodologia SCRUM para o desenvolvimento, de forma que todos os dias eram reportados ao SCRUM *Master* o andamento da pesquisa e do desenvolvimento. Ao final de cada *sprint*, era realizada uma reunião com o gerente do projeto a fim de definir as próximas etapas da pesquisa.

4.1 Investigação dos Parâmetros do Inversor

Por meio de consulta aos manuais do inversor de frequência, foi possível listar os seus parâmetros. A lista de parâmetros do inversor que podem ser consultadas em tempo real por meio do visor é apresentada na Tabela 4, junto com suas respectivas descrições.

Tabela 4 – Parâmetros possíveis de acessar via visor do inversor.

Variável	**Descrição**
Pac	Potência AC do inversor, em watts.
Qac	Potência relativa AC do inversor, em volt-ampères reativos.
Sac	Potência aparente AC do inversor, em volt-ampères.
CosPhi	É o cosseno do ângulo de defasagem existente entre a tensão e a corrente gerada pelo inversor. O cosseno de ϕ pode ser positivo ou negativo: Positivo: o inversor injeta energia reativa positiva. A corrente está adiantada em relação à tensão. Negativo: o inversor injeta energia reativa negativa. A corrente está atrasada em relação à tensão.
Vac	Tensão AC do inversor, em volts.
Iac	Corrente AC através do inversor, em ampères.
Fac	Frequência gerada pelo inversor, em hertz.
Pdc	Potência DC do inversor, em watts.
Pdc1	Potência DC do string 1, em watts.
Pdc2	Potência DC do string 2, em watts.
Riso	Resistência de isolamento do campo solar, em quilo-ohms.
Vdc1	Tensão de entrada DC do string 1.
Vdc2	Tensão de entrada DC do string 2.
Idc1	Corrente de entrada DC do string 1.
Idc2	Corrente de entrada DC do string 2.
Total	
Energia	Energia total injetada pelo inversor em toda a sua vida útil.
Tempo	Tempo total que o inversor esteve injetando na rede.
Conexões	Número total de conexões com a rede.
Parcial	
Energia	Energia parcial injetada pelo inversor.
Tempo	Tempo parcial que o inversor esteve conectado com a rede.
Conexões	Número parcial de conexões na rede.
Pac	Potência AC, em watts.
Pwatt	Potência injetada na rede pública no modo autoconsumo, em watts.

Alguns dos parâmetros que podem ser acessados e modificados por meio dos comandos *Modbus* são detalhados na próxima sessão.

4.2 Investigação do Comandos do Inversor

A unidade de comandos do inversor especifica os comandos que são necessários para implementar a comunicação entre um cliente *Modbus* e o inversor, tanto para a leitura quanto para a escrita nos registradores. A interação se dá por meio do protocolo *Modbus*, em que o inversor é o servidor e um cliente faz requisições a fim de escrever ou ler em algum registrador e o inversor

responde de acordo.

Na Tabela 5 são apresentados alguns dos parâmetros e os respectivos endereços dos registradores e limites de valores.

Tabela 5 – Alguns dos parâmetros do inversor.

Parâmetro	**Endereço**	**Valor Mín.**	**Valor Máx.**	**Tipo**
Comando	1000	0	32	Leitura/Escrita
Data 1	1001	$-2^{15}-1$	$2^{15}-1$	Leitura/Escrita
Data 2	1002	$-2^{15}-1$	$2^{15}-1$	Leitura/Escrita
Valor tan(ϕ) de referência	1006	$-2^{15}-1$	$2^{15}-1$	Leitura
Valor cos(ϕ) de referência	1008	$-2^{15}-1$	$2^{15}-1$	Leitura
Modo de operação do inversor	1011	0	4	Leitura

Para enviar comandos ao inversor via *Modbus* é necessário definir os valores dos registradores *Data 1*, *Data 2* e, por fim, o valor do código de comando (*cmd*), apresentados na Tabela 6.

Tabela 6 – Endereços dos registradores responsáveis por receber o comando e os dados.

Parâmetro	**Endereço**	**Tipo**
Comando	1000 (0x03E8)	Leitura/Escrita
Data 1	1001 (0x03E9)	Leitura/Escrita
Data 2	1002 (0x03EA)	Leitura/Escrita

Para ler e escrever nos registradores, é possível utilizar algumas das funções do protocolo *Modbus* apresentadas na Tabela 1. Como apresentado na seção 3.2, é possível ler o conteúdo de registradores utilizando a função 0x03, escrever em um registrador utilizando a função 0x06 ou escrever em múltiplos registradores utilizando a função 0x10.

Algumas das ações possíveis que o cliente pode requisitar do inversor são apresentadas na Tabela 7. Cada uma das ações tem um código de comando (*cmd*) associado. Para alguns comandos, é necessário definir o valor dos campos *Data 1* e *Data 2* (*e.g.* comandos para definir variáveis associadas à bateria), alguns somente do campo *Data 1* (*e.g.* definir tan(ϕ)). Há também comandos que não fazem uso dos campos *Data 1* e *Data 2* (*e.g.* parar inversor, iniciar inversor). Os valores definidos nos campos *cmd*, *Data 1* e *Data 2* precisam todos estar no formato hexadecimal para serem enviados.

Tabela 7 – Lista de comandos.

cmd	Comando	*Data 1*	Mín	Máx
0x1	Definir tan(ϕ) de referência	tan(ϕ) vezes $2^{15}-1$	-0.75 (-24876)	0.75 (24876)
0x5	Parar inversor	Não utilizado	-	-
0x6	Iniciar inversor	Não utilizado	-	-
0x7	Colocar inversor em standby	Não utilizado	-	-
0x8	Definir potência ativa de referência	Porcentagem da potência ativa vezes $2^{15}-1$	-1.0 (327676)	1.0 (327676)
0x9	Definir potência reativa de referência	Porcentagem da potência ativa vezes $2^{15}-1$	-1.0 (327676)	1.0 (327676)
0xA	Definir cos(ϕ) de referência	cos(ϕ) vezes $2^{15}-1$	-0.80 (-26214)	0.80 (26214)

4.3 Exemplos de Comando para o Inversor

São apresentados alguns exemplos a seguir de como definir os parâmetros e enviar comandos para o inversor usando *Modbus*.

4.3.1 Tangente de *Phi*

Para mudar a tangente de phi (*cmd* 0x01) para o valor máximo de 0,75 usando a função 0x06 é necessário definir primeiramente o valor correspondente dado por 0,75*32767 = 24757 = 0x5FFF e escrever no campo *Data 1*. O valor enviado precisa estar em hexadecimal. Após definir o valor de *Data 1*, é enviado o código de comando que é 0x0001, nesse caso. Na Tabela 8 é apresentado os *frames* enviados via *Modbus* e as respectivas respostas esperadas do inversor.

Tabela 8 – Frame de dados para definir a tan(ϕ) como 0.75 utilizando a função 0x06.

	Frame *Modbus* (Função 0x06)			
	ID Cliente	Função	Endereço	Dados
1	01	06	03 E9 (*Data 1*)	5F FF
2	01	06	03 E8 (*cmd*)	00 01
Resposta				
1	01 06 03 E9 5F FF			
2	01 06 03 E8 00 01			

Para mudar a tangente de phi para o valor máximo de 0,75 usando a função 0x10, pode-se definir os valores de *cmd* e *Data 1* no mesmo comando. Transformando 0,75 no valor adequado por meio de 0,75*32767 = 24757 = 0x5FFF e sabendo que o código de comando é 0x0001, então pode-se observar as mensagens enviadas e respostas esperadas do inversor na Tabela 9.

Tabela 9 – Frame de dados para definir a tan(ϕ) como 0.75 utilizando a função 0x10.

	Frame *Modbus* (Função 0x10)						
	ID Cliente	Função	Endereço do primeiro registrador	Núm. de Registradores	Núm. de Bytes	*cmd*	*Data 1*
1	01	10	03E8	00 02	0004	0001	5F FF
Resposta							
1	01 10 03 E8 00 01						

Com a função 0x10 é necessário definir o endereço do primeiro registrador, o número de registradores, o tamanho em bytes desses registradores seguido dos valores a serem escritos. Os valores informados serão escritos a partir do endereço do primeiro registrador, então, no exemplo da Tabela 9, 0x0001 é escrito no registrador 0x03E8 e 0x5FFF é escrito no registrador 0x03E9.

4.3.2 Parar Inversor

Para parar o inversor, basta definir o comando (*cmd* 0x05), não havendo necessidade de definir valor de *Data 1* ou *Data 2*, como apresentado na Tabela 10. Cabe observar que nesse caso a função 0x10 foi utilizada para escrever em somente um registrador.

Tabela 10 – Frame de dados para parar o inversor.

	Frame *Modbus* (Função 0x10)					
	ID Cliente	Função	Endereço do primeiro registrador	Núm. de Registradores	Núm. de Bytes	*cmd*
1	01	10	03E8	00 01	0002	0005
Resposta						
1	01 10 03 E8 00 01					

4.3.3 Iniciar Inversor

Para iniciar o inversor, basta definir o comando (*cmd* 0x06), não havendo necessidade de definir valor de *Data 1* ou *Data 2*, sendo o *frame* utilizando a função 0x10 apresentado na Tabela 11.

Tabela 11 – Frame de dados para iniciar o inversor.

	Frame *Modbus* (Função 0x10)					
	ID Cliente	Função	Endereço do primeiro registrador	Núm. de Registradores	Núm. de Bytes	*cmd*
1	01	10	03E8	00 01	0002	0006
Resposta						
1	01 10 03 E8 00 01					

4.3.4 Inversor em *Standby*

Para colocar o inversor em *standby*, basta definir o comando (*cmd* 0x07), não havendo necessidade de definir valor de *Data 1* ou *Data 2*. O *frame* usando a função 0x10 é apresentado na Tabela 12.

Tabela 12 – Frame de dados para definir modo standby do inversor.

	Frame *Modbus* (Função 0x10)					
	ID Cliente	Função	Endereço do primeiro registrador	Núm. de Registradores	Núm. de Bytes	*cmd*
1	01	10	03E8	00 01	0002	0007
Resposta						
1	01 10 03 E8 00 02					

4.3.5 Cosseno de *Phi*

Para definir o cosseno de de phi (*cmd* 0x0A) para o valor mínimo permitido de -0,80 é necessário fazer a conversão para o valor adequado por meio de -0,80*32767 = -26213 = 0x999B e escrever o *cmd*. O *frame* sando a função 0x10 nesse caso é apresentado na Tabela 13.

Tabela 13 – Frame de dados para definir a cos(ϕ) como -0.8.

	Frame *Modbus* (Função 0x10)						
	ID Cliente	Função	Endereço do primeiro registrador	Núm. de Registradores	Núm. de Bytes	*cmd*	*Data 1*
1	01	10	03E8	00 02	0004	000A	99 9B
Resposta							
1	01 10 03 E8 00 02						

4.3.6 Parâmetros da Bateria

Para controlar parâmetros da bateria (*cmd* 0x1A), é necessário definir o comando, *Data 1* e *Data 2*. Em *Data 1* é armazenada a informação de qual o parâmetro que de deseja modificar e *Data 2* o valor do parâmetro.

Tabela 14 – Comandos para definir parâmetros da bateria no inversor.

Ação (Ex. de valor)	***Data 1***	***Data 2***
Corrente de carga (10 A)	0x00	0x000A (10)
Corrente de descarga (50 A)	0x01	0x0032 (50)
Tensão de carga (140 V)	0x02	0x008C (140)
Potência do grid (3000 W)	0x0A	0x0BB8 (3000)
Tensão de descarga (100 V)	0x0B	0x0064 (100)
Tensão da bateria x10 (120,5 V)	0x0C	0x04B5 (1205)
Corrente da bateria (20 A)	0x0D	0x0014 (20)
Temperatura da bateria x10 ($21,8^{\circ}$)	0x0E	0x00DA (218)
Tensão da bateria x100 (48,53 V)	0x0F	0x12F5 (4853)

Para definir, por exemplo, a corrente da bateria como 20 A, é necessário definir o comando como 0x1A, *Data 1* 0x000D e *Data 2* como 20 = 0x0014. Desta forma, o *frame Modbus* enviado e a resposta são apresentados na Tabela 15.

Tabela 15 – Frame de dados para definir a corrente da bateria como 20 A.

	Frame *Modbus* (Função 0x10)							
	ID Cliente	Função	Endereço do primeiro registrador	Núm. de Registradores	Núm. de Bytes	*cmd*	*Data 1*	*Data 2*
1	01	10	03E8	00 03	0006	00 1A	00 0D	00 14
Resposta								
1	01 10 03 E8 00 03							

Os *frames Modbus* apresentados podem ser utilizados para verificar se a comunicação entre o servidor e o cliente está ocorrendo adequadamente. Sabendo o código da função e os limites dos valores dos dados é possível gerar os *frames*.

4.4 Desenvolvimento do Protótipo de Servidor *Modbus*

Devido à pandemia, não foi possível realizar os testes de comunicação com o inversor de frequência. Desta forma, foi criado um servidor síncrono *Modbus* do tipo TCP usando o *PyModbus* com 2000 registradores definidos pelo contexto para simular o inversor. O código implementado é apresentado no Algoritmo 4.1.

```
1 '''
2 Servidor utilizando o PyModbus
3
4 sync_server.py
5 '''
6
```

```
# Importa módulos necessários
from pymodbus.server.sync import StartTcpServer
from pymodbus.device import ModbusDeviceIdentification
from pymodbus.datastore import ModbusSequentialDataBlock
from pymodbus.datastore import ModbusSlaveContext, ModbusServerContext
from pymodbus.transaction import ModbusRtuFramer

# Inicializa armazenamento de dados com 2000 endereços sequenciais
store = ModbusSlaveContext(
    di = ModbusSequentialDataBlock(0, [0]*2000),
    co = ModbusSequentialDataBlock(0, [0]*2000),
    hr = ModbusSequentialDataBlock(0, [0]*2000),
    ir = ModbusSequentialDataBlock(0, [0]*2000)
)
# Define contexto
context = ModbusServerContext(slaves=store, single=True)
# Inicializa o servidor no localhost e porta 5020
StartTcpServer(context, identity=identity, address=("localhost", 5020))
```

Algoritmo 4.1 – Servidor *Modbus*.

Por simplicidade, preferiu-se implementar uma versão mais básica do servidor usando o *PyModbus* porque o comportamento alvo é a comunicação que poderia ser verificada por meio dos *frames Modbus* trocados entre cliente e servidor. O servidor também poderia ser do tipo UDP dado que não é uma aplicação de risco, bastando importar e utilizar a função *StartUdpServer* ao invés da função *StartTcpServer*.

4.5 Desenvolvimento do Protótipo de Cliente *Modbus*

O protótipo do cliente para se comunicar com o inversor é apresentado no Algoritmo 4.2. Para exemplificar seu funcionamento, são apresentadas 3 funções, uma para iniciar o inversor (*start_inversor()*), uma para definir o valor da tan(ϕ) (*set_tan_phi()*), uma para parar o inversor (*stop_inversor()*).

A função *write_register()* é responsável por escrever um determinado valor no endereço informado como parâmetro. Nessa função, é utilizada o método do *PyModbus write_registers()* que implementa a função 0x10 do protocolo *Modbus*. Foi optado por utilizá-la para escrever em um registrador por vez por motivos de simplicidade e reutilização da função.

```
'''
Protótipo de Cliente do inversor

client_modbus.py
'''

```

```
7  # Importa bibliotecas necessárias
8  from pymodbus.client.sync import ModbusTcpClient as ModbusClient
9  from pymodbus.compat import iteritems
10 from collections import OrderedDict
11
12 def tohex(value, bits=16):
13     ''' Essa função retorna o valor em hexadecimal a partir
14     de um valor em decimal possivelmente negativo.
15
16     A função hex() do python não converte adequadamente
17     quando o parâmetro é negativo
18
19     Parameters
20     ----------
21     value : int
22         Um valor em decimal entre -2**(bits-1) e 2**(bits-1)
23     bits: int, opcional
24         Número de bits para representar a conversão
25     Returns
26     -------
27     : int
28         Valor em hexadecimal
29     '''
30     return hex((value & 2**(bits-1)-1))
31
32 def write_register(reg_add, value, unit=0x01, ip='127.0.0.1', port=5020):
33     ''' Essa função escreve um dado valor em determinado registrador
34     do servidor modbus
35
36     Parameters
37     ----------
38     reg_add : int
39         Endereço do registrador (começando do 0x0)
40     value : int
41         Valor do parâmetro
42     unit : int, opcional
43         Id do client modbus
44     ip: str, opcional
45         Endereço IP do server modbus
46     port: int, opcional
47         Porta do server modbus
48     Returns
49     -------
50     None
51     '''
52
53     # Inicia conexão com o servidor
```

```
54      client = ModbusClient(ip, port=port)
55      client.connect()
56      # Escreve valor no endereço informado menos 1000 (começa no zero na biblioteca)
57      rq = client.write_registers(reg_add, [value], unit=unit)
58      rr = client.read_holding_registers(reg_add, 1, unit=unit)
59
60      try:
61          # Verifica se houve de erro
62          assert(rq.function_code < 0x80)
63          # verifica se o valor escrito é o desejado
64          assert(rr.registers == [value])
65          log.debug("Resultado: %s"%hex(rr.registers[0]))
66      except Exception as e:
67          log.error("Houve um erro durante a escrita do valor" + \
68              " no endereço %s: %s."%(reg_add, e))
69      client.close()
70
71  def set_tan_phi(value):
72      ''' Escreve nos registradores o valor para a tangente de Phi
73      Referência
74
75      Parameters
76      ----------
77      value : float
78          Um valor entre -0.75 e 0.75
79      Returns
80      -------
81      None
82      '''
83      if(value >= -0.75 and value <= 0.75):
84          data = int(value * (2**15-1))
85          data = int(tohex(data), 16)
86          log.debug("Escrevendo tan(phi) %s => %s"%(value, hex(data)))
87          # Escreve Data 1 no endereço 0x03E9
88          address = 0x03E9
89          # Escreve no registrador o valor da tangente de phi
90          write_register(address, data)
91          # Escreve cmd no endereço 0x03E8
92          address = 0x03E8
93          cmd = 0x0001
94          # Escreve comando no registrador
95          write_register(address, cmd)
96      else:
97          log.error("O valor especificado está fora" + \
98              "do intervalo [-0.75, 0.75]")
99
100 def start_inversor():
```

```
101     ''' Essa função inicializa o inversor
102     '''
103     reg_add = 0x03E8
104     cmd = 0x0006
105     log.debug("Iniciando inversor")
106     write_register(reg_add, cmd)
107 
108 def stop_inversor():
109     ''' Essa função para o inversor
110     '''
111     reg_add = 0x03E8
112     cmd = 0x0005
113     log.debug("Parando inversor")
114     write_register(reg_add, cmd)
115 
116 if __name__ == "__main__":
117     start_inversor()
118     set_tan_phi(0.75)
119     stop_inversor()
```

Algoritmo 4.2 – Protótipo do Cliente *Modbus*.

As respostas são recebidas automaticamente pela biblioteca *PyModbus*, assim como a geração e verificação dos valores de CRC (*checksum*) dos *frames*. Foi necessário implementar uma função de conversão de números em decimal para hexadecimal porque alguns parâmetros podem ser negativos e a função *hex()* do *Python* convertia para um valor diferente do esperado para o envio.

4.6 Testes com o Protótipo do Cliente *Modbus*

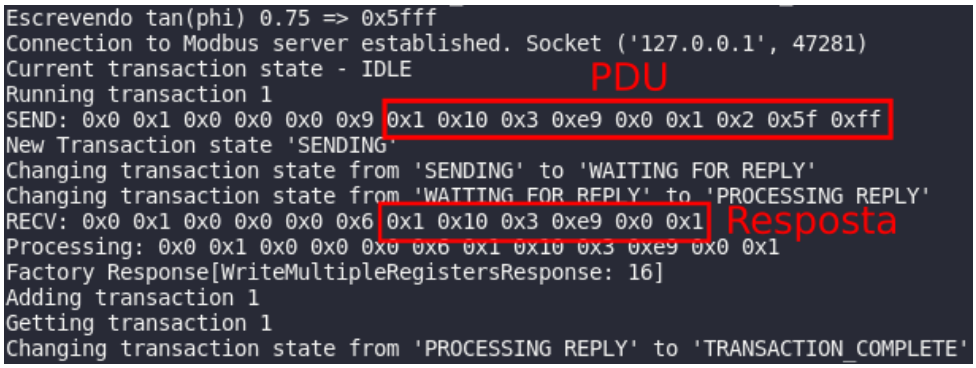
O primeiro teste se refere a definição da tangente de phi como 0,75 utilizando a função 0x10 que é a utilizada pelo cliente desenvolvido. Os *frames* necessários para isso são apresentados na Tabela 16.

Tabela 16 – Frame de dados para definir a tangente de phi como 0,75 utilizando a função 0x10.

	Frame *Modbus* (Função 0x10)					
	ID Cliente	Função	Endereço do primeiro registrador	Núm. de Registradores	Núm. de Bytes	*cmd*
1	01	10	03E9	00 01	0002	5FFF
2	01	10	03E8	00 01	0002	0001
Resposta						
1	01 10 03 E9 00 01					
2	01 10 03 E8 00 01					

Ao executar o servidor desenvolvido e, posteriormente, executar o cliente, as mensagens referentes ao envio do primeiro *frame* são apresentadas na Figura 7.

Figura 7 – Representação das mensagens trocadas entre cliente e servidor durante o envio do comando para definir a tangente de phi como 0,75.

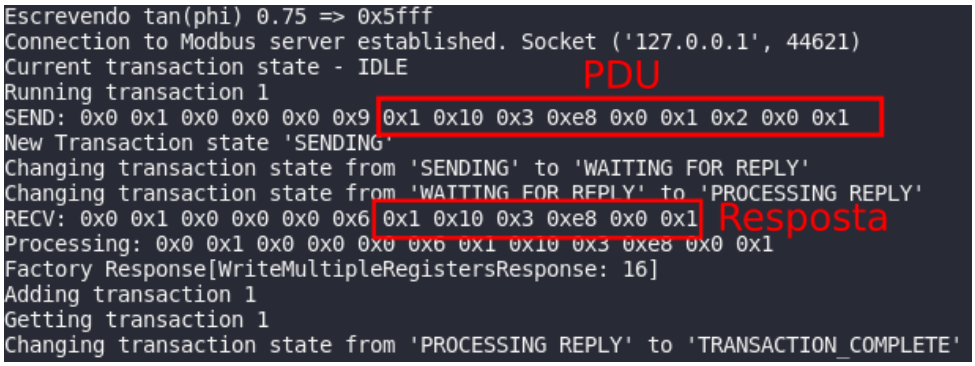


Fonte: Autoria Própria.

O primeiro *frame* que é referente à definição de *Data 1* como 0x5FFF, pode ser observado na Figura 7 como 0x0 0x1 0x0 0x0 0x0 0x9 0x1 0x10 0x3 0xe9 0x0 0x1 0x2 0x5f 0xff. Vale lembrar que foi utilizado o *Modbus* TCP, de forma que 0x0 0x1 0x0 0x0 0x0 0x9 constitui o cabeçário MBAP e 0x1 0x10 0x3 0xe9 0x0 0x1 0x2 0x5f 0xff é o *frame* esperado (PDU). A resposta dada pelo servidor está no *frame* 0x0 0x1 0x0 0x0 0x0 0x6 0x1 0x10 0x3 0xe9 0x0 0x1, em que 0x0 0x1 0x0 0x0 0x0 0x6 é o cabeçário MBAP e 0x1 0x10 0x3 0xe9 0x0 0x1 é como o *frame* esperado.

Já as mensagens trocadas para o segundo *frame* que é referente à definição do comando (*cmd*) como 0x0001 é apresentado na Figura 8.

Figura 8 – Representação das mensagens trocadas entre cliente e servidor durante o envio do comando para definir o comando para tangente de phi.



Fonte: Autoria Própria.

Como pode ser observado na Figura 8, o *frame* enviado pelo cliente é 0x0 0x1 0x0 0x0 0x0 0x9 0x1 0x10 0x3 0xe8 0x0 0x1 0x2 0x0 0x1 que é constituído pelo cabeçario MBAP e o *frame* esperado. A resposta do servidor é 0x0 0x1 0x0 0x0 0x0 0x6 0x1 0x10 0x3 0xe8 0x0 0x1 que também é constituída do cabeçário MBAP e o *frame* esperado.

Os testes foram repetidos para as funções *start_inversor()* e *stop_inversor()* e os resultados são apresentados na Tabela 17 e na Tabela 18. As respostas obtidas para os dois casos foi 0x0 0x1 0x0 0x0 0x0 0x6 0x1 0x10 0x3 0xe8 0x0 0x1.

Tabela 17 – Frame de dados para iniciar o inversor.

	Frame *Modbus* (Função 0x10)					
	ID Cliente	Função	Endereço do primeiro registrador	Núm. de Registradores	Núm. de Bytes	*cmd*
1	01	10	03E8	00 01	0002	0006
Frame Enviado						
1	0x0 0x1 0x0 0x0 0x0 0x9 0x1 0x10 0x3 0xe8 0x0 0x1 0x2 0x0 0x6					

Tabela 18 – Frame de dados para parar o inversor.

	Frame *Modbus* (Função 0x10)					
	ID Cliente	Função	Endereço do primeiro registrador	Núm. de Registradores	Núm. de Bytes	*cmd*
1	01	10	03E8	00 01	0002	0005
Frame Enviado						
1	0x0 0x1 0x0 0x0 0x0 0x9 0x1 0x10 0x3 0xe8 0x0 0x1 0x2 0x0 0x5					

Neste capítulo foram apresentadas as atividades desenvolvidas pelo aluno durante o período de estágio. As atividades envolveram a investigação acerca do inversor de frequência e implementação de um cliente *Modbus* para realizar a comunicação com o inversor. No próximo capítulo serão apresentadas as considerações finais do trabalho.

5 Considerações Finais

Neste relatório foram descritas as principais atividades desenvolvidas em um projeto de pesquisa e desenvolvimento como parte da disciplina de Estágio Supervisionado no Laboratório de Sistemas Embarcados e Computação Pervasiva (Embedded) da UFCG. O aluno desenvolveu relacionadas à investigação e implementação de um cliente *Modbus* em *Python* para se comunicar com um inversor de frequência. Foi cumprida a carga horária de aproximadamente 214 horas, ultrapassando a carga horária mínima para o estágio supervisionado matriculado (180 horas).

Foram investigados os parâmetros do inversor passíveis de acesso e modificação via *Modbus* e os comandos necessários para o acesso e modificação. Devido à pandemia, não foi possível realizar os testes de comunicação com o inversor, sendo necessário implementar um servidor *Modbus* para fazer o papel do inversor.

Foram realizados testes de comunicação entre o cliente e o servidor e verificou-se que foi possível realizar a comunicação com sucesso, além de definir adequadamente os valores nos registradores do servidor após o envio dos comandos via *Modbus*. Isso pôde ser observado por meio da comparação entre os *frames Modbus* trocados entre cliente e servidor com os *frames Modbus* indicados na unidade de especificação dos comandos do inversor de frequência.

Foi possível colocar em prática vários conceitos de engenharia adquiridos no decorrer do curso e desenvolver o trabalho em equipe, atividades de planejamento, de pesquisa, de desenvolvimento e de produção de relatórios técnicos com os resultados obtidos. Além disso, foi necessário entrar em contato com novos assuntos e tecnologias , complementando a formação do aluno ao longo do estágio.

Referências

[1] Embedded Lab. **LABORATÓRIO DE SISTEMAS EMBARCADOS E COMPUTAÇÃO PERVASIVA**. Disponível em: <https://www.embedded.ufcg.edu.br/>. Acesso em: outubro de 2019.

[2] Energy Sage. **String inverters vs. power optimizers vs. microinverters**. Disponível em: <https://www.energysage.com/solar/101/string-inverters-microinverters-power-optimizers/>. Acesso em: 15 de Maio de 2021.

[3] Modbus. **Modbus application protocol specification V.1.1b3**. Disponível em: <https://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf>. Acesso em: 15 de Maio de 2021.

[4] Modbus. **Modbus Messaging Implementation Guide V.1.0b**. Disponível em: <https://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf>.Acesso em: 15 de Maio de 2021.

[5] PyModbus. Disponível em: <https://pymodbus.readthedocs.io/en/latest/readme.html>. Acesso em: 15 de Maio de 2021.