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

SISTEMA DE SUPERVISÃO E AQUISIÇÃO DE DADOS COM INTERFACE WEB APLICADO À MICROGERAÇÃO DE ENERGIA HIDRÁULICA

TRABALHO DE CONCLUSÃO DE CURSO

VANESSA FURTADO DE LIMA

Santa Maria, RS, Brasil

2016

SISTEMA DE SUPERVISÃO E AQUISIÇÃO DE DADOS COM INTERFACE WEB APLICADO À MICROGERAÇÃO DE ENERGIA HIDRÁULICA

por

Vanessa Furtado de Lima

Monografia apresentada ao Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Bacharel em Engenharia de Controle e Automação.

Orientador: Prof. Dr. Robinson Figueiredo de Camargo

Santa Maria, RS, Brasil
2016

Endereço: Universidade Federal de Santa Maria, RS, Brasil.
Fone (055) 55 84154527; End. Eletr: vanessalima.fd@gmail.com

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Engenharia de Controle e Automação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Conclusão de Curso

SISTEMA DE SUPERVISÃO E AQUISIÇÃO DE DADOS COM INTERFACE WEB APLICADO À MICROGERAÇÃO DE ENERGIA HIDRÁULICA

elaborada por
Vanessa Furtado de Lima

como requisito parcial para obtenção do grau de
Bacharel em Engenharia de Controle e Automação

COMISSÃO EXAMINADORA

_____
Robinson Figueiredo de Camargo, Dr. (UFSM)
(Presidente/Orientador)

_____
Humberto Pinheiro, Ph.D. (UFSM)

_____
Jorge Rodrigo Massing, Dr. (UFSM)

Santa Maria, 06 de Janeiro de 2016.

AGRADECIMENTOS

Agradeço, primeiramente, aos meus pais por sempre buscarem o melhor pra mim e com isso me auxiliaram e me motivaram incansavelmente durante meu período de graduação. Meus pais me ensinaram a valorizar o estudo e a educação desde pequena, me incentivando a buscar conhecimento e realizar meus objetivos.

Também quero ressaltar a importância do meu orientador, Prof. Dr. Robinson Camargo, que sempre me auxiliou e me guiou, transmitindo conhecimentos que foram importantes não só para este trabalho, mas também para a minha jornada acadêmica.

Agradeço também aos meus amigos e amigas que me proporcionaram momentos marcantes durante a graduação e que sempre estiveram presentes para me ajudar e incentivar durante os períodos mais difíceis.

Por fim, agradecer aos colegas do Grupo de Eletrônica de Potência e Controle que estão sempre dispostos a ajudar, tirar dúvidas e ensinar. Foi no GEPOC onde comecei a colocar em prática os conhecimentos adquiridos em sala de aula, o que foi muito importante para meu crescimento acadêmico.

"Se as coisas são inatingíveis... ora!
Não é motivo para não querê-las...
Que tristes os caminhos, se não fora
A presença distante das estrelas!"

Mário Quintana

RESUMO

Trabalho de Conclusão de Curso
Curso de Engenharia de Controle e Automação
Universidade Federal de Santa Maria

SISTEMA DE SUPERVISÃO E AQUISIÇÃO DE DADOS COM INTERFACE WEB APLICADO À MICROGERAÇÃO DE ENERGIA HIDRÁULICA

AUTORA: VANESSA FURTADO DE LIMA
ORIENTADOR: ROBINSON FIGUEIREDO DE CAMARGO
Data e Local da Defesa: Santa Maria, 06 de Janeiro de 2016

Com o recente aumento de interesse no tema de geração distribuída, a micro geração de energia hidráulica, mesmo já desenvolvida nos primórdios da geração de energia elétrica, tornou-se novamente interessante objeto de estudo. Sendo assim, este trabalho propõe o desenvolvimento de um sistema de supervisão e aquisição de dados aplicado a micro geração de energia hidráulica. Por ser livre de taxas e se adequar facilmente a diversos meios físicos, o protocolo de comunicação *Modbus* TCP/IP foi implementado utilizando o *kit* de desenvolvimento F28M36x *Concerto* para troca de dados com o sistema de supervisão desenvolvido. O *kit* de desenvolvimento possui um DSP multi núcleos de excelente desempenho onde também são desenvolvidas aplicações para aquisição e tratamento de dados das placas de instrumentação do sistema de micro geração de energia hidráulica. Através do sistema de supervisão é possível visualizar os dados do sistema em tempo real com uma interface amigável utilizando um *driver Modbus*. Os dados adquiridos pelo sistema de supervisão são armazenados em um banco de dados de um servidor *Web*. É possível monitorar as variáveis do sistema remotamente a partir de uma interface *Web* que foi desenvolvida utilizando as linguagens de programação PHP e HTML.

Palavras-chave: Sistema Supervisório, *Modbus*, Sistemas embarcados, Interface *Web*, Micro Geração de Energia Hidráulica.

ABSTRACT

Bachelor Final Project
Bachelor of Control and Automation Engineering
Federal University of Santa Maria

SUPERVISORY CONTROL AND DATA ACQUISITION SYSTEM WITH WEB INTERFACE APLYED TO MICRO HYDROPOWER PLANTS

AUTHOR: VANESSA FURTADO DE LIMA
ADVISER: ROBINSON FIGUEIREDO DE CAMARGO
Defense Place and Date: Santa Maria, January 06th, 2016

Micro hydropower plants have become an interesting object of study because the interest in distributed generation has increased recently, even though it had already been developed in the early days of electricity generation. Therefore, this project proposes the development of a supervisory control and data acquisition system applied to micro hydropower plants. Because it is free of charges and able to easily adapt to several physical interfaces, Modbus TCP/IP protocol was implemented using Concerto F28M36x Experimenter's Kit to exchange data with the supervisory system. Concerto F28M36x Experimenter's Kit includes a multi-core DSP with excellent performance where applications are developed for acquisition and processing of data from system's instrumentation boards. The supervisory system with a Modbus driver allows real-time monitoring of the transmitted data with a friendly interface. Also, a web server database stores data acquired by the supervisory system. It is possible to monitor the micro hydropower plant system remotely using a web interface that was developed using PHP and HTML programming languages.

Key words: Supervisory System, Modbus, Embedded Systems, Web Interface, Micro Hydropower Plants.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE APÊNDICES

SUMÁRIO

INTRODUÇÃO

O interesse em geração distribuída e sistemas de geração que utilizam energia renovável, tais como energia hidráulica, eólica, solar e biomassa, vêm crescendo consideravelmente nas últimas décadas. Existem necessidades que precisam de atualização, pois não são mais atendidas pelo sistema atual, como questões de âmbito econômico, tecnológico, social e ambiental. Então destacam-se inúmeros fatores que contribuem para o interesse nestas fontes de energia, dentre estes citam-se (MOHD et al, 2008):

- A crescente demanda de energia por parte dos países desenvolvidos e em desenvolvimento;
- Provável escassez futura de combustíveis fósseis;
- Problemas com o gerenciamento de resíduos provenientes de energia nuclear;
- Escassez de recursos para a construção de grandes centrais de geração e redes de distribuição por parte dos países emergentes;
- Insuficiência de geração de energia por parte de alguns países desenvolvidos;
- Crescente preocupação com a emissão de poluentes e as alterações climáticas.

Em âmbito nacional, os debates acerca da geração de energia elétrica chegam a um consenso em que, ao longo dos anos, a evolução demográfica e o crescimento da atividade econômica tem resultado num constante aumento do consumo de energia elétrica no país (ANEEL, 2014).

O Brasil dispõe de uma matriz elétrica de origem predominantemente renovável, com destaque para a geração hidráulica que responde por 65,2% da oferta interna, segundo o Balanço Energético Nacional (EPE, 2015). Além disso, em todo o mundo, o Brasil é o país com maior potencial hidrelétrico: um total de 260 mil MW, segundo o Plano 2015 da Eletrobrás. Destes, pouco mais de 30% se transformaram em usinas construídas ou outorgadas, demonstrando que muito ainda pode ser feito para expandir o parque hidrelétrico (ANEEL, 2008).

Diante desse quadro, é preciso pensar em alternativas que respondam à necessidade de expansão e diversificação do parque gerador elétrico do país. Neste sentido, pesquisas vêm sendo desenvolvidas buscando a expansão dos sistemas de geração de energia descentralizados ou a utilização de sistemas distribuídos. Sendo assim, uma das possibilidades de geração distribuída é o uso de micro centrais hidroelétricas para geração de energia.

Para que seja possível prever e controlar a geração de energia da planta é necessário modelar todos os componentes para simular e emular o funcionamento dos sistemas de microgeração de energia hidráulica. Logo, para o controle e supervisão deste tipo de central geradora é importante o desenvolvimento e aprimoramento dos sistemas de supervisão deste processo.

Os sistemas de supervisão proporcionam o monitoramento, operação, controle e aquisição de dados em tempo real. Os dados adquiridos são apresentados na tela de um supervisório ao operador, permitindo que este monitore as variáveis adquiridas do processo e proceda de forma eficiente diante de diversas situações. Esse tipo de sistema também tem capacidade de proporcionar um histórico de dados, que pode ser utilizado para analise de desempenho do processo monitorado.

Para que o interfaceamento da planta hidroelétrica com um sistema supervisório permita o monitoramento da mesma, é necessária a utilização de uma rede de comunicação entre a central geradora e o sistema de supervisão.

Assim, com a necessidade da troca de dados entre computadores e controladores lógico programáveis surgiram as redes de comunicação industrial, permitindo o compartilhamento de recursos e bases de dados com segurança aos usuários da informação.

Entre as redes industriais mais utilizadas hoje tem-se o protocolo de comunicação *Modbus*. Esse protocolo foi criado na década de 1970 e é um dos mais antigos e até hoje é um dos mais utilizados protocolos de comunicação, principalmente em sistemas de automação industrial (NI, 2014). A utilização do protocolo *Modbus* para comunicação entre a central geradora e o sistema de supervisão torna-se importante devido ao sucesso do protocolo, que é livre de taxas de licenciamento e se adapta facilmente a diversos meios físicos, sendo uma solução de baixo custo considerando o ramo de automação industrial.

Finalmente, o presente trabalho propõe o desenvolvimento de um sistema de supervisão para microgeração de energia hidráulica utilizando o protocolo de comunicação *Modbus*, juntamente da criação de uma interface de monitoramento via *Web*. Para implementação desse projeto será utilizado um sistema que emula o funcionamento de uma microgeração de energia hidráulica presente no laboratório do Grupo de Eletrônica de Potência e Controle (GEPOC), também um DSP da série *Concerto* que controla e adquire dados do sistema e embarca o protocolo de comunicação.

CAPÍTULO 1 REVISÃO BIBLIOGRÁFICA

1.1 Sistemas de Supervisão

Sistemas de supervisão e controle industrial, também conhecidos no ambiente industrial como sistemas SCADA (*Supervisory Control and Data Acquisition*), permitem o monitoramento em tempo real dos sinais representativos de medidas e do estado atual do processo industrial, recolhendo esses dados de ambientes complexos e apresentando de modo amigável para o operador através de interface humano-máquina (ROSÁRIO, 2005).

No início os sistemas supervisórios eram arquiteturas centralizadas e sem conectividade externa, que empregavam *hardware* e *software* proprietários. As informações eram visualizadas em um painel de lâmpadas e indicadores, sem qualquer interface amigável ao operador. Entretanto, atualmente os sistemas de manufatura utilizam tecnologias de computação e de comunicação que permitem automatizar a monitoração e controle de processos, aumentando assim a confiabilidade, flexibilidade e conectividade desses sistemas. A partir disso, as informações podem ser coletadas por equipamentos de aquisição de dados, analisadas e disponibilizadas de forma compreensível, com recursos gráficos e interatividade, ao operador. A tecnologia atual também permite que operadores e administradores do processo em ambientes afastados geograficamente possuam acesso remoto simultâneo do sistema de monitoração do processo (SANTOS, 2007).

Basicamente, os sistemas SCADA podem ser visto como um conjunto de *software* e *hardware* destinado à aquisição e transmissão de dados de um processo. O objetivo do sistema de supervisão é informar o operador do atual estado de funcionamento dos equipamentos e dos processos monitorados, gerenciando assim o processo. Em outras palavras, as principais funcionalidades do sistema SCADA são automaticamente realizar a leitura das variáveis em tempo real, manter a comunicação entre o sistema e o operador, criar uma base de dados do sistema monitorado e o acionamento de alarmes perante eventos de determinada importância (DUMITRU e GLIGOR, 2012).

Os sistemas SCADA são melhores aplicados em processos que são relativamente simples de controlar e monitorar e que necessitam de uma intervenção frequente ou imediata. Alguns dos processos que são ideais para utilizar um sistema de supervisão são (BOYER, 2004):

- Pequenas estações de geração de energia hidrelétrica que estão localizadas em locais remotos;
- Estações de produção de óleo e gás;
- Sistemas de transmissão elétrica;
- Sistemas de irrigação.

É possível também elencar algumas vantagens proporcionadas pelo uso de sistemas SCADA (VIANNA, 2008):

- Redução de gastos com montagem de painéis de controle e projeto;
- Redução de custos da aquisição de instrumento de painel, pois no sistema SCADA eles são virtuais;
- Redução de espaço necessário para a sala de controle;
- Dados disponíveis em formato eletrônico, facilitando a geração de relatórios e a integração com sistemas ERP (*Enterprise Resource Planning*) ou SIGE (Sistemas Integrados de Gestão Empresarial);
- Praticidade de operação, pois o monitoramento e controle do processo são mostrados de forma amigável ao operador.

Além disso, pode-se dizer que o sistema supervisório é composto por quatro elementos: processo físico, *hardware* de controle, *software* de supervisão e rede de comunicação (DANEELS e SALTER, 1999).

1.1.1 Processo Físico

O processo físico é o elemento que se deseja monitorar e/ou controlar. As informações deste elemento são capturadas através de instrumentos, tanto para o controle do processo quanto para a gerência de dados.

Os sensores relacionam informações sobre uma grandeza física que precisa ser mensurada, tais como, temperatura, velocidade, corrente, aceleração, posição etc. Então esses elementos são responsáveis pela leitura do estado em que o processo se encontra.

Já os atuadores são dispositivos que modificam uma variável controlada. Recebem um sinal proveniente do controlador e agem sobre o sistema controlado.

1.1.2 *Hardware* de Controle

O processo de controle e aquisição de dados se inicia nas estações remotas, PLCs (*Programmable Logic Controllers*) e RTUs (*Remote Terminal Units*), com a leitura de valores atuais dos dispositivos que a ele estão associados e seu respectivo controle. Os PLCs e RTUs são unidades computacionais específicas, utilizadas nas instalações fabris (ou qualquer outro tipo de instalação que se deseja monitorar) para a funcionalidade de ler entradas, realizar cálculos ou controles e atualizar saídas. A diferença entre os PLCs e as RTUs é que os primeiros possuem mais flexibilidade na linguagem de programação e controle das entradas e saídas, enquanto as RTUs possuem uma arquitetura mais distribuída entre sua unidade de processamento central e os cartões de entradas e saídas (SILVA e SALVADOR, 2011).

1.1.3 Rede de Comunicação

A rede de comunicação é a plataforma por onde as informações fluem dos PLCs/RTUs para o sistema SCADA. A integração de informações por meio de uma rede de comunicação entre os mais diversos níveis hierárquicos numa indústria possibilita a redução dos custos, e uma melhor eficiência na tomada de decisão (SOUZA *et al*, 2006).

Geralmente, uma rede de comunicação constitui-se de duas sub-redes, como mostrado na Figura 1, que são denominadas:

- Rede de campo, que é a responsável pela aquisição dos dados do processo, podendo este estar em locais remotos;
- Rede local de supervisão, que disponibiliza e compartilha os dados do processo em uma LAN (*Local Area Network*) para que usuários possam requisitar esses dados e utilizá-los para o controle e supervisão do processo.

As redes de campo geralmente utilizam o mecanismo de comunicação mestre/escravo com o intuito de uma comunicação determinística. Nesta sub-rede, as RTUs desempenham a função de estações escravas, respondendo somente às solicitações feitas pela unidade central mestre. A unidade mestre é quem inicia o processo de troca de informações, podendo endereçar uma mensagem para um único escravo ou para vários escravos ao mesmo tempo. Assim, a estação escrava pode fornecer os dados solicitados pela unidade mestre ou realizar uma ação requisitada na transação.

Já em relação a rede local de supervisão, esta geralmente utiliza o mecanismo de comunicação cliente/servidor, onde clientes distribuídos numa rede realizam a requisição de dados a uma unidade servidora, então esta coleta as informações requisitadas e as repassa ao cliente que as solicitou.

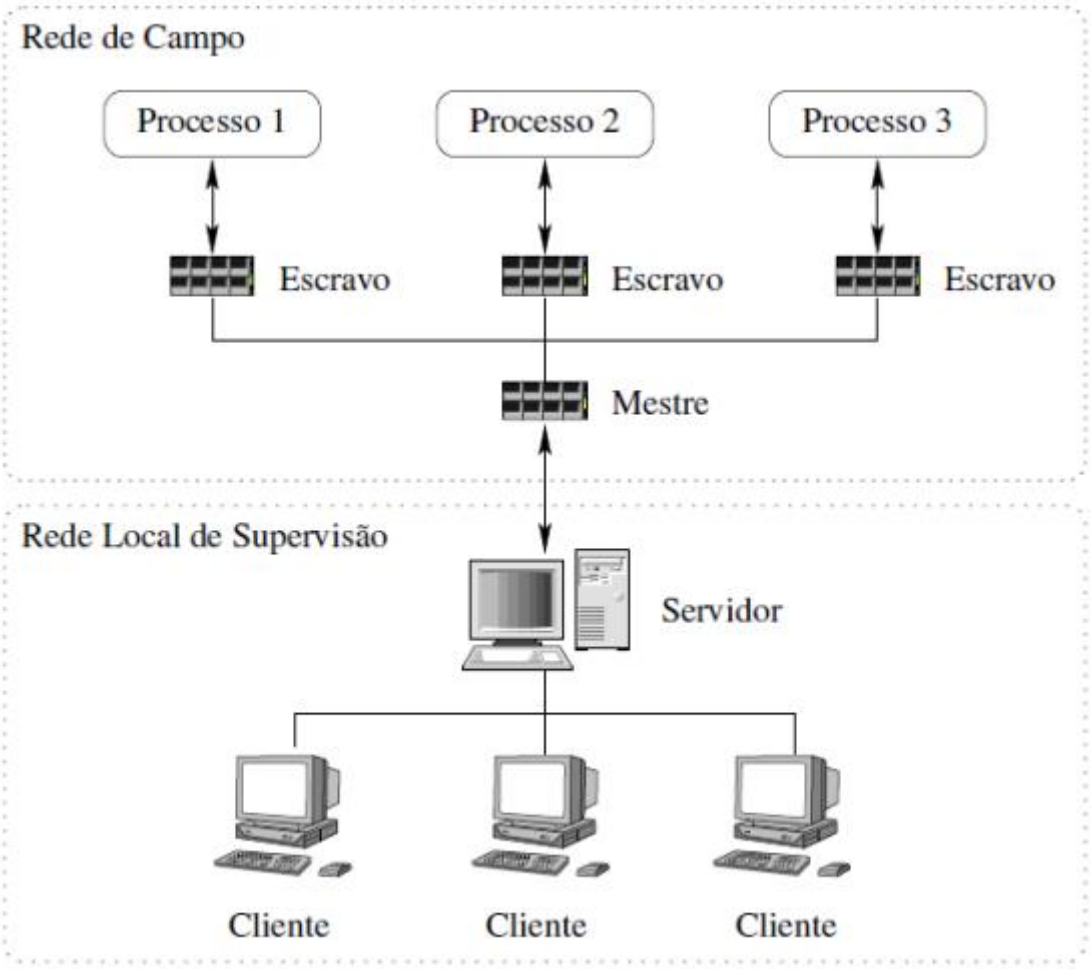


Figura 1 – Estrutura de uma Rede de Comunicação.
Fonte: (SOUZA, 2005)

1.1.4 *Software* de Supervisão

Os *softwares* de supervisão são instalados em servidores centrais e/ou clientes distribuídos. São responsáveis pela obtenção dos dados do processo em tempo real, através da comunicação com os *hardwares* de controle, e dispõem de uma interface que permite o usuário a inspeção, monitoração e operação de um processo físico, tornando o processamento das variáveis de campo mais rápido e eficiente.

As telas de supervisão, *drivers* de comunicação, gráficos informativos, alarmes eventuais, histórico de dados, relatórios e variáveis auxiliares desses *softwares* são

configurados pelo desenvolvedor (programador) através de uma plataforma de desenvolvimento de aplicações de supervisão. Portanto, através da configuração do *software*, o operador pode selecionar a tela que deseja utilizar e assim visualizar os valores numéricos das variáveis monitoradas do processo.

Um *software* de supervisão deve possuir algumas funções básicas, entre elas (SILVA, 2009):

- Apresentação das variáveis monitoradas em tempo real;
- Gráfico de tendências das variáveis do processo;
- Alarmes com descrição e codificação de cores;
- Sinalização visual e sonora de alarmes;
- Sinalização do *status* de operação dos equipamentos;
- Modificação dos parâmetros operacionais;
- Armazenamentos de eventos;
- Armazenamento de histórico;
- Telas de acompanhamento operacional;
- Gerar relatórios.

Atualmente existem diversos ambientes de desenvolvimento de sistemas supervisórios comerciais, podendo destacar os seguintes:

1.1.4.1 LabView

Este *software* é um ambiente de desenvolvimento para aquisição de sinal, análise de medidas e apresentação de dados. Apresenta um grande número de cartões para a aquisição de dados de campo e facilidade para construir aplicativos em um ambiente totalmente gráfico. O LabView é um *software* da *National Instruments*.

1.1.4.2 Elipse SCADA

O Elipse SCADA é um *software* para desenvolvimento de sistemas de supervisão. Totalmente configurável pelo usuário, permite a monitoração de variáveis em tempo real, através de gráficos e objetos que estão relacionados com as variáveis físicas de campo. Também é possível fazer acionamentos e enviar ou receber informações para equipamentos de aquisição de dados (ELIPSE SCADA, 2004).

O *software* possui conexão ODBC (*Open Data Base Connectivity*) que permite troca de dados com uma base de dados, incorpora a tecnologia OPC (*Object Linking and Embedding for Process Control*) para aquisição de dados de CLPs e troca de informações com outros aplicativos SCADA em tempo real e emite relatórios, históricos e controle de alarmes.

1.1.4.3 InTouch

O *software* InTouch é uma ferramenta de desenvolvimento de sistemas de supervisão e controle, permitindo a criação e execução de aplicativos com interfaces. Possui uma série de funcionalidades que servem para monitorar e operar qualquer sistema parcialmente ou completamente automatizado. A monitoração ocorre por meio de telas gráficas, com indicações dinâmicas do estado dos equipamentos e grandezas analisadas. Possui também ferramentas para realizar gráficos de tendência, relatórios e histórico de eventos e de processo (SA, 2014).

1.1.4.4 Elipse E3

O Elipse E3 é um sistema de supervisão e controle de processos desenvolvido para atender atuais requisitos de conectividade, flexibilidade e confiabilidade, sendo ideal para uso em sistemas críticos. O software oferece uma plataforma de rápido desenvolvimento de aplicações, alta capacidade de comunicação e garantia de expansão. A solução permite a comunicação com inúmeros protocolos e equipamentos (ELIPSE SOFTWARE, 2015).

1.2 Redes industriais

A utilização das redes industriais permite a comunicação rápida e confiável entre equipamentos e o uso de mecanismos padronizados, que são hoje em dia, fatores indispensáveis no conceito de produtividade industrial.

A introdução das redes no ambiente industrial por sinais elétricos analógicos aconteceu a partir da década de 1960 e permitiu a substituição de grande quantidade de tubos utilizados para a transmissão pneumática, o que reduziu substancialmente o custo de

instalação dos sistemas, bem como o tempo de transmissão dos sinais, naturalmente lento nos sistemas pneumáticos (GUTIERREZ e PAN, 2008).

Como os sistemas de automação industrial tornam-se cada vez maiores e o número de dispositivos de automação só aumentava, tornou-se muito importante para a automação industrial que padrões fossem criados para que fosse possível interconectar diferentes dispositivos de automação de um jeito padronizado. Um considerável esforço internacional para a padronização tem acontecido no que se trata das redes locais. O padrão OSI (Open Systems Interconnection) permite que dois dispositivos de automação se comuniquem de forma confiável independente do fabricante (DJIEV, 2003).

1.3 Padrão ISO/OSI

Devido à proliferação das redes proprietárias, a *International Organization for Standardization* - ISO definiu um modelo de referência para a interconexão de sistemas abertos em 1978, o RM-OSI. O padrão OSI segue a filosofia das arquiteturas multicamadas, gerencia estruturação da comunicação de dados, através das sete camadas mostradas pela Figura 2.

Figura 2 – As sete camadas do padrão OSI

Cada camada tem um objetivo definido e comunica-se com as camadas adjacentes através de uma interface, que define as operações elementares e os serviços que a camada inferior oferece à camada considerada. Quando o padrão OSI foi desenvolvido foram levados

em consideração alguns princípios para determinar quantas camadas o modelo deveria ter. Estes princípios são:

- Cada camada corresponde a um nível de abstração necessário no modelo;
- Cada camada possui suas funções próprias e bem definidas;
- As funções de cada camada foram escolhidas segundo a definição dos protocolos padronizados internacionalmente;
- A escolha da fronteira entre cada camada deveria ser definida de modo a minimizar o fluxo de informação nas interfaces;
- O número de camadas deveria ser suficientemente grande para evitar a realização de funções muito diversas por uma mesma camada;
- O número de camadas deveria ser suficientemente pequeno para evitar uma alta complexidade da arquitetura (REYNDERS, MACKAY e WRIGHT, 2005).

Resumidamente as principais funções das camadas do modelo OSI são:

- Camada de Aplicação

  Faz o interfaceamento entre os protocolos de comunicação e o aplicativo que solicito ou receberá a informação pela rede.

- Camada de Apresentação

  Assume funções associadas à formatação, sintaxe e semântica dos dados transmitidos que permitirá a interpretação correta da informação pelo receptor.

- Camada de Sessão

  Controla a comunicação entre os usuários. É responsável por agrupar as mensagens, coordenar a transferência de dados entre as camadas e sincronizar o diálogo.

- Camada de Transporte

  Gerencia a comunicação entre dois sistemas, ou seja, conexões fim a fim.

- Camada de Rede

  Tem a função de determinar os caminhos (roteamento) e endereçamento lógico.

- Camada de Enlace de Dados

  Realiza a montagem e envio de um quadro de dados de um sistema para outro.

- Camada Física

  Tem como função definir como e com que padrão os dados serão enviados e recebidos no meio físico do canal de comunicação.

1.4 Modelo TCP/IP

O acrônimo TCP significa protocolo de controle de transmissão (*Transmission Control Protocol*) o que coloca sua implementação na camada de transporte do modelo OSI. Já, o acrônimo IP significa protocolo de inter-redes (*Internet Protocol*) o que coloca sua implementação na camada de rede do modelo OSI. O TCP/IP representa, de certa maneira, o conjunto das regras de comunicação na Internet e baseia-se na noção de endereçamento IP, isto é, o fato de fornecer um endereço IP a cada máquina da rede a fim de poder encaminhar pacotes de dados.

Embora possa se situar tanto o TCP quanto o IP nas camadas do modelo OSI, o modelo TCP/IP não se baseia no modelo OSI e foi concebido apenas na prática sem uma estruturação teórica.

Este modelo foi pela primeira vez definido por Cerf e Kahn em 1974 e surgiu da necessidade de uma rede que "sobrevivesse" (seguisse funcionando) a eventuais perdas de hardware de sub-rede mantendo-se em atividade (com conexões ativas). Essa necessidade levou a escolha de uma rede de comutação (troca) de pacotes baseada em uma camada de interligação de redes sem conexões. Esta camada é chamada de inter-redes (Internet ou Internetwork) e integra toda arquitetura de rede numa grande rede de redes locais (SCHAF, 2012).

Conforme a Figura 3, cada camada do modelo TCP/IP corresponde a uma ou mais camadas do modelo de referência de sete camadas de interconexão dos sistemas abertos (OSI), proposto pela *International Standards Organization* (ISO).

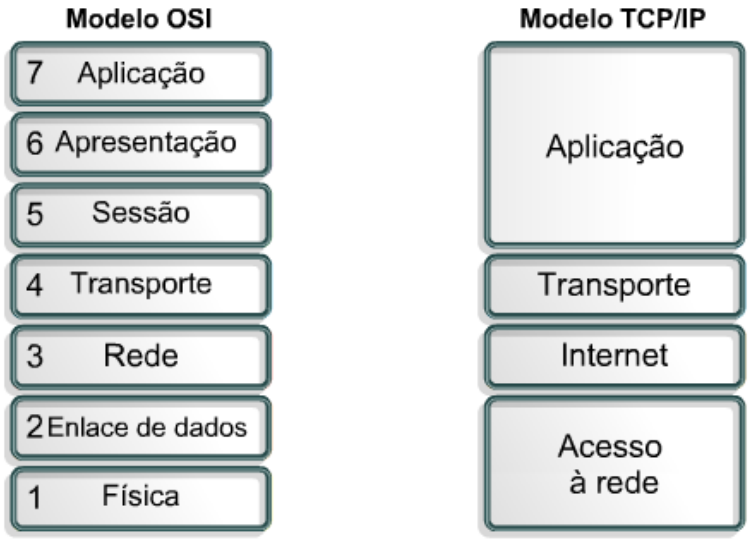


Figura 3 – Comparação entre o modelo OSI e TCP/IP.
Fonte: (SCHAF, 2012)

Os tipos de serviços executados em cada camada do modelo TCP/IP são descritos com mais detalhes abaixo:

- Camada de Aplicação

  Em contraposição ao modelo OSI, não foram implementadas camadas de sessão e apresentação no modelo TCP/IP. Esta camada compreende todos os protocolos de alto nível como: HTTP (*Hypertext Transfer Protocol*), POP (*Post Office Protocol*), DNS (*Domain Name System*), SMTP (*Simple Mail Transfer Protocol*), etc.

- Camada de Transporte

  Embora na camada de transporte do modelo TCP/IP seja implícito que seja usado o TCP, também pode ser usado o protocolo UDP (*User Datagram Protocol*). O TCP é usado para serviços orientados a conexões, confiáveis, com ordenamento e controle de fluxo. Já o UDP é exatamente o contrário, é usado para serviços sem conexão, não confiável, sem ordenação nem controle de fluxo.

- Camada de Internet

  Esta camada tem a tarefa de permitir que *hosts* transmitam pacotes em qualquer rede e garante que os pacotes trafeguem independentemente até o destino. O formato do pacote oficial é o IP. O roteamento e controle de congestionamento também são uma tarefa desta camada.

- Camada de Acesso à Rede

  Esta camada também é chamada de camada de *host* ou de rede. O modelo TCP/IP não especifica muito bem esta camada, exceto que deve suprir conexão à rede para possibilitar a entrega de pacotes IP. Normalmente é utilizado a padronização *Ethernet* (IEEE 802.3) ou *WiFi* (IEEE 802.11) nesta camada.

1.5 Protocolo *Modbus*

Desenvolvido e publicado pela Modicon Industrial Automation Systems em 1979 para uso do seu CLP, tornou-se um padrão de fato na indústria. É um dos mais antigos protocolos utilizados em redes de controladores lógicos programáveis para aquisição de sinais de instrumentos e comandar atuadores. Atualmente parte do grupo Schneider Electric, a Modicon colocou as especificações e normas que definem o *Modbus* em domínio público. Por esta razão é utilizado em milhares de equipamentos existentes e é uma das soluções de rede mais baratas a serem utilizadas em automação industrial.

A tecnologia de comunicação no protocolo é o mestre-escravo, conforme Figura 4. A comunicação é sempre iniciada pelo mestre, e os nós escravos não se comunicam entre si. O mestre pode transmitir dois tipos de mensagens aos escravos, dentro de uma mesma rede:

- Mensagem tipo *unicast*: o mestre envia uma requisição para um escravo definido e este retorna uma mensagem-resposta ao mestre. Portanto, nesse modo são enviadas duas mensagens: uma requisição e uma resposta;
- Mensagem tipo *broadcast*: o mestre envia a requisição para todos os escravos, e não é enviada nenhuma resposta para o mestre (MORAIS e CASTRUCCI, 2007).

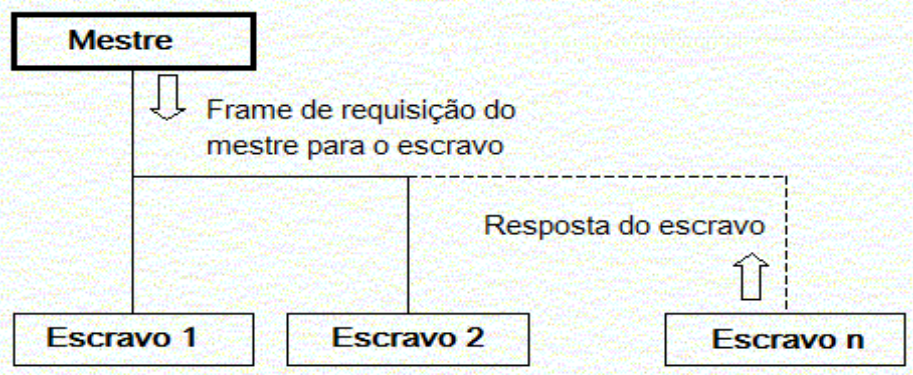


Figura 4 – Comunicação mestre e escravo.
Fonte: (PACONTROL, 2014)

O mestre possui várias atribuições, são elas:

- Assegurar a troca de informações entre as ECL (Estações de Controle Local) ou ETD (Equipamento Terminal de Dados - RTU).
- Assegurar um diálogo com outros mestres ou com um computador (gestão centralizada do conjunto do processo).
- Assegurar a programação ou passagem de parâmetros para os escravos.

1.5.1 Descrição do protocolo

O protocolo *Modbus* define uma única PDU (*Protocol Data Unit*), independente do meio físico. O mapeamento (encapsulamento) do protocolo *Modbus* em um barramento ou rede específica introduz alguns campos adicionais, criando a ADU (*Application Data Unit*) como mostra a Figura 5. Este ADU é o *frame Modbus*.

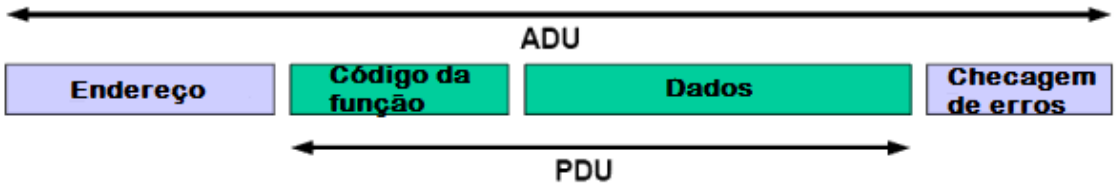


Figura 5 – *Frame Modbus*

As mensagens são codificadas e constituídas por um conjunto de caracteres hexadecimais ou ASCII. O tamanho máximo da PDU é de 253 bytes, então:

- RS232/RS485 ADU = 253 (dados) + 1 (endereço) + 2 (CRC - *Cyclic Redundancy Check*)
- TCP/IP ADU = 253 (dados) + 7 (MBAP – *Modbus Application Protocol*)

Os serviços são especificados por códigos de função com cada serviço possuindo um formato de mensagem para a requisição e outro para a resposta.

Os códigos de funções válidos vão de 1 a 255, sendo que de 128 a 255 são reservados para respostas de exceção (erro). O bit mais significativo é o que decide o tipo do código. Códigos de sub-função podem ser adicionados aos códigos de função para definir múltiplas ações (SCHAF, 2012).

1.5.2 Meio Físico

O *Modbus* não especifica o meio de comunicação (meio físico) e sim a estrutura dos quadros, usualmente funciona sobre: RS-232, RS-485 e *Ethernet*, mas pode também funcionar sobre RF (rádio frequência) e Fibra Ótica. No *Modbus* os dois meios físicos mais usuais são *Modbus* Padrão e *Modbus* TCP/IP, detalhados a seguir (SCHAF, 2012).

1.5.2.1 *Modbus* Padrão

Utilizado para comunicação dos CLPs com os módulos de I/O, atuadores de válvulas, transdutores de energia, etc. Neste padrão a transmissão é serial assíncrona sobre vários meios: EIA/TIA-232-E (RS-232), EIA/TIA-422 (RS-422), EIA/TIA-485-A (RS-485), fibra óptica, RF. Naturalmente que neste caso sempre são usadas as especificações elétricas, ópticas ou de RF destes meios, assim como as velocidades compatíveis com estes meios (SCHAF, 2012).

1.5.2.2 *Modbus* TCP/IP

Usado para comunicação entre sistemas de supervisão e CLPs. Os dados, em formato binário, são encapsulados em quadros *Ethernet*, pacotes IP e segmentos TCP. Utiliza a porta 502 da pilha TCP/IP em velocidade compatível com o *Ethernet* (tipicamente 100 Mbps).

A função primária do TCP é garantir que todos os pacotes de dados sejam recebidos corretamente, enquanto o IP garante que a mensagem seja enviada corretamente ao destinatário. Assim, nota-se que a combinação TCP/IP é apenas um protocolo de transporte, o qual não define os dados a serem enviados ou como serão interpretados, este é o trabalho do protocolo *Modbus*.

Essencialmente, a mensagem transmitida através do protocolo *Modbus* TCP/IP é simplesmente uma comunicação *Modbus* encapsulada em um pacote *Ethernet* TCP/IP. O modelo de mensagem *Modbus* TCP/IP tem um cabeçalho específico chamado MBAP (*Modbus Application Protocol*), como mostrado na Figura 6.

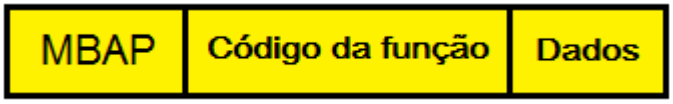


Figura 6 – Modelo de Mensagem Modbus TCP/IP

O cabeçalho MBAP tem tamanho de 7 bytes, e é composto pelos seguintes campos:

- *Transaction identifier*: usado para identificação da resposta para a transação (2 bytes);
- *Protocol identifier*: 0 (zero) indica *Modbus* (2 bytes);
- *Length*: contagem de todos os próximos bytes (2 bytes);
- *Unit identifier*: utilizado para identificar o escravo remoto em uma rede *Modbus* RTU (1 byte).

O *Modbus* TCP/IP não acrescenta ao quadro um campo de checagem de erros, entretanto o *frame ethernet* já utiliza CRC-32 tornando desnecessário outro campo de checagem. O cliente *Modbus* TCP/IP deve iniciar uma conexão TCP com o servidor a fim de enviar as requisições. A porta TCP 502 é a porta padrão para conexão com servidores *Modbus* TCP/IP (FREITAS, 2015).

1.5.3 Modos de Codificação

Existem dois modos de codificação de mensagens:

- *Modbus* RTU - os bytes são transmitidos de forma binária pura. A checagem da mensagem é com um campo de CRC de 16 bits (2 bytes).
- *Modbus* ASCII - cada byte é dividido em dois de representação hexadecimal e codificação em ASCII (7 bits). Nota-se que esta codificação é um desperdício de bits. A checagem é por LRC (*Longitudinal Redundancy Check*).

1.5.4 Códigos das Funções

É onde o mestre especifica o tipo de serviço ou função solicitada ao escravo (leitura, escrita, etc). No protocolo *Modbus*, cada função é utilizada para acessar um tipo específico de dado. Abaixo, na Tabela 1, são mostradas as funções do protocolo.

Tabela 1 – Funções do Protoloco Modbus

Código da função	Descrição
1	Leitura de bloco de bits do tipo *coil* (saída discreta).
2	Leitura de bloco de bits do tipo *input* (entradas discretas).

3	Leitura de bloco de registradores do tipo *holding*.
4	Leitura de bloco de registradores do tipo *input*.
5	Escrita em um único bit do tipo *coil* (saída discreta).
6	Escrita em um único registrador do tipo *holding*.
15	Escrita em bloco de bits do tipo *coil* (saída discreta).
16	Escrita em bloco de registradores do tipo *holding*.

1.5.5 Troca de Mensagens entre Mestre e Escravo

As trocas de mensagens entre mestre e escravo seguem uma estrutura bem simples representada na Figura 7 – Estrutura de transação mestre e escravo abaixo. O endereço no caso é sempre do escravo e quem faz a solicitação é somente o mestre, por isso não há necessidade de inclusão do endereço do mestre (SCHAF, 2012).

Figura 7 – Estrutura de transação mestre e escravo.
Fonte: (SCHAF, 2012)

O campo de dados da mensagem enviada de um mestre para um escravo contém informações adicionais que auxiliam o escravo a executar a ação requerida no campo "código da função" (*function code*), como:

- Endereços dos registradores (registro inicial);
- Quantidade de registros a serem lidos;
- Contador da quantidade de *bytes* no campo de dados;
- O campo de dados pode não existir. Neste caso, o próprio código da função sozinho especifica a ação requerida;
- Se não ocorrer nenhum erro na função especificada na requisição, a resposta do escravo conterá o dado requisitado, caso contrário o campo dados conterá um código de exceção.

O formato da requisição (*request*) do mestre segue o seguinte padrão:

- Nº do endereço do escravo (1 *byte*)
- Código da função a realizar (1 *byte*)

  Comandos de escrita ou leitura
- Dados

  Endereço da posição de memória (2 *bytes*)

  Quantidade de operandos (2 *bytes*)

  Dados a serem escritos no escravo (até 250 *bytes*)
- Controle de erros (2 *bytes*): CRC-16

O formato da resposta (*response* ou *reply*) do escravo segue o seguinte padrão:

- Nº do endereço do escravo (1 *byte*)
- Código da função realizada (1 *byte*)

  Comando solicitado de escrita ou leitura
- Dados

  Quantidade de dados da resposta (1 *bytes*)

  Dados solicitados para o escravo (até 250 *bytes*)
- Controle de erros (2 *bytes*): CRC-16

1.6 Padrão OPC

O padrão OPC (*Object Linking and Embedding for Process Control*) é usado para compatibilizar os protocolos da camada de aplicação, a fim de evitar que diversos *drivers* tivessem que ser criados para permitir a comunicação entre dois dispositivos. Esse padrão foi inicialmente liderado pela *Microsoft* e especificado pela OPC *Foundation*.

A introdução de uma interface padronizada fez com que os fabricantes de *hardware* reduzissem a quantidade de *drivers* desenvolvidos e implementassem o OPC *Server* (Servidor OPC – responsável por disponibilizar os dados). Da mesma forma, os fabricantes de software passaram a utilizar a interface do OPC *Client* (Cliente OPC – responsável por requisitar os dados).

As principais vantagens do emprego de uma comunicação OPC são (FONSECA, 2002):

- Padronização das interfaces de comunicação entre os servidores e clientes de dados de tempo real, facilitando a integração e manutenção dos sistemas;
- Eliminação da necessidade de *drivers* de comunicação específicos;
- Melhoria do desempenho e otimização da comunicação entre dispositivos de automação;
- Interoperabilidade entre sistemas de diversos fabricantes;
- Redução dos custos e tempo para desenvolvimento de interfaces e *drivers* de comunicação, com consequente redução do custo de integração de sistemas;
- Facilidade de desenvolvimento e manutenção de sistemas e produtos para comunicação em tempo real.

CAPÍTULO 2 OBJETIVOS

2.1 Objetivos Gerais

Desenvolvimento de um Sistema de Supervisão e Aquisição de Dados (SCADA) aplicado a microgeração de energia hidráulica utilizando o protocolo de comunicação *Modbus* TCP/IP. Além disso, a criação de uma interface de monitoramento do sistema via *Web*.

2.2 Objetivos Específicos

- Implementar a comunicação *Modbus* TCP/IP em um DSP da série *Concerto*.
- Desenvolvimento de uma placa para conectar o DSP na placa de interface presente no sistema que emula a microgeração de energia hidráulica.
- Implementação da comunicação *Modbus* TCP/IP entre o DSP do sistema que emula a microgeração de energia hidráulica e o computador.
- Utilização de um ambiente de simulação (*software*) do sistema de microgeração de energia hidráulica para obtenção de resultados de simulação.
- Aquisição de grandezas elétricas que está sendo gerada pelo sistema de microgeração com a utilização do DSP.
- Desenvolvimento de um sistema supervisório.
- Realizar a conexão do sistema de supervisão com um banco de dados.
- Integração do sistema supervisório com o DSP utilizando o protocolo *Modbus* TCP/IP.
- Desenvolvimento de uma interface de monitoramento via web.
- Obtenção de resultados experimentais e comparação com os resultados de simulação obtidos.

CAPÍTULO 3 MATERIAIS E MÉTODOS

3.1 Sistema de Micro Geração Hidráulica

O sistema de microgeração de energia hidráulica utilizado neste trabalho foi desenvolvido nos laboratórios do Grupo de Eletrônica de Potência e Controle da Universidade Federal de Santa Maria (UFSM). Este sistema emula uma turbina hidráulica a partir de um inversor de frequência e um motor elétrico, conforme Figura 8. Este conjunto é acoplado a um gerador de indução auto-escitado (GIAE). Este tipo de gerador foi escolhido para o projeto, pois possui autoproteção contra sobrecarga e requer baixa manutenção, conferindo confiabilidade ao sistema (MARRA e POMILIO, 2000; RAI *et al*, 1993; YOUSSEF *et al*, 2008). A topologia escolhida para geração de energia emprega um gerador assíncrono que requer uma rotação constante, o que implica ter um método de controlar a velocidade da turbina (SCHERER e DE CAMARGO, 2011). O diagrama esquemático presente na Figura 9 apresenta o sistema de emulação da micro geração hidráulica.

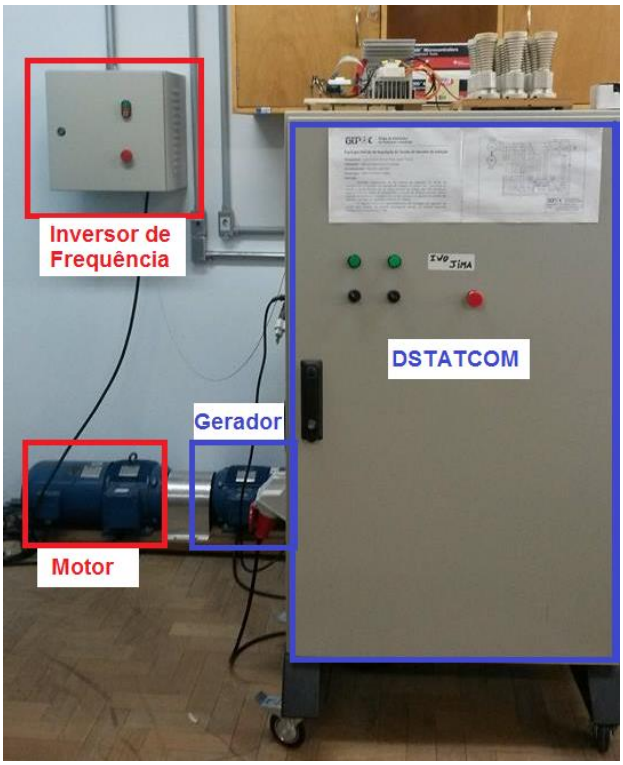


Figura 8 – Protótipo de micro geração de energia hidráulica.

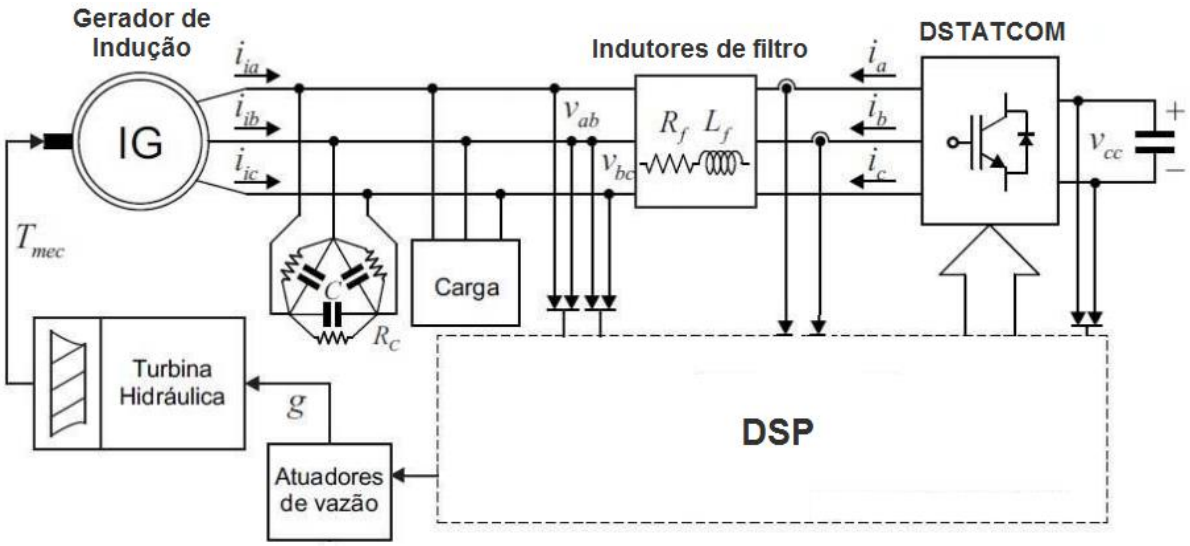


Figura 9 – Diagrama esquemático do sistema de micro geração de energia hidráulica.

Para o interfaceamento, aquisição e condicionamento dos diferentes sinais do sistema são utilizadas placas de circuitos eletrônicos. As seguintes placas são utilizadas no sistema de micro geração:

- Placa de interface com DSP;
- Placa de interface com o inversor;
- Placa de aquisição de tensão;
- Placa de aquisição de corrente;
- Placa de condicionamento de sinais;
- Placa de acionamento de contatoras.

3.2 Sistema Proposto de Supervisão e Aquisição de Dados

A arquitetura geral do sistema proposto para supervisão e aquisição de dados pode ser visto na Figura 10. A arquitetura do sistema basicamente consiste no DSP concerto F28M36x adquirindo dados do sistema de micro geração de energia hidráulica e enviando-os em tempo real para um roteador utilizando o padrão *Ethernet* através do protocolo *Modbus* TCP/IP. Um computador com um sistema de supervisão recebe os dados enviados pelo roteador por uma rede *wireless* (sem fio). Os dados recebidos pelo sistema de supervisão são armazenados em um banco de dados na *Web*, com isso, através de uma página hospedada na *Web* é possível monitorar em tempo real os dados adquiridos.

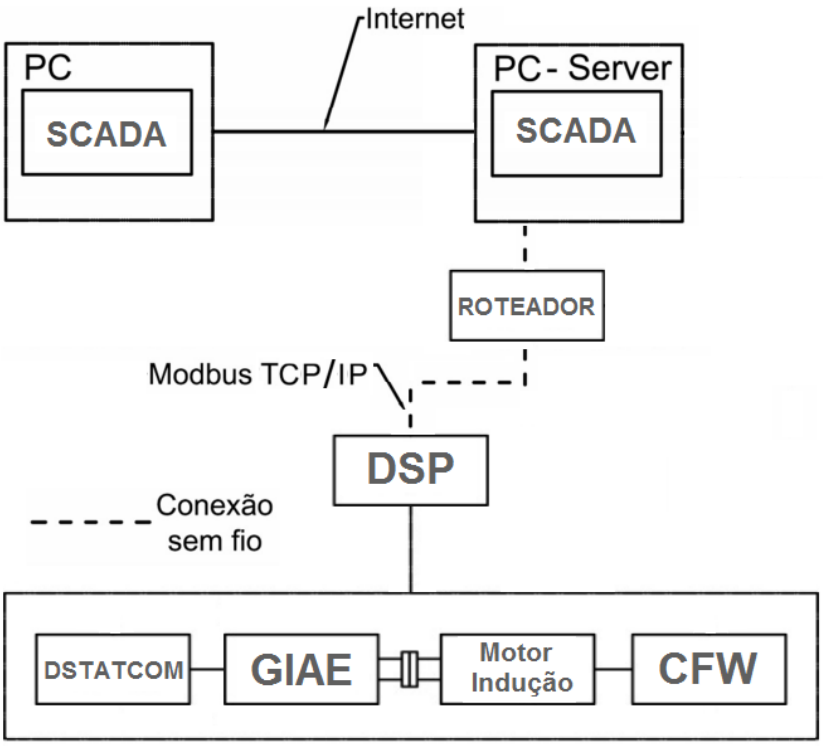


Figura 10 – Visão geral do sistema proposto de supervisão e aquisição de dados.

3.2.1 *Kit* de Desenvolvimento Concerto F28M36x

O Concerto é uma família de microcontroladores multi núcleos com subsistemas de comunicação independente e de controle em tempo real. O subsistema de comunicação é baseado na CPU ARM Cortex-M3 de 32 bits incluindo uma grande variedade de periféricos para comunicação como Ethernet 1588, USB, CAN, UART, I²C entre outras. Já o subsistema de controle em tempo real é baseado na CP C28x de 32 bits, ponto flutuante, incluindo periféricos com flexibilidade e precisão, por exemplo, ePWM, com proteções contra faltas. (TEXAS INSTRUMENTS, 2012). É possível visualizar o *kit* de desenvolvimento através da Figura 11.

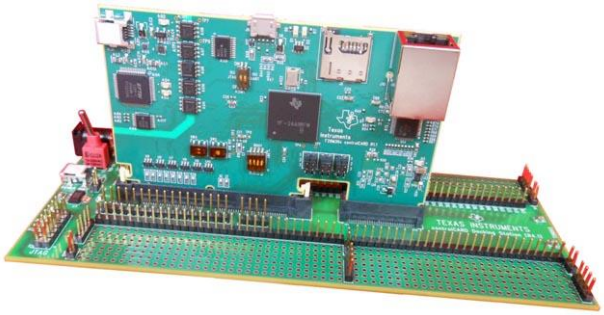


Figura 11 – *Kit* de Desenvolvimento Concerto F28M36x.
Fonte: (RL, 2012)

3.2.2 *Software* Code Composer Studio 6

É um ambiente de desenvolvimento integrado que suporta o *kit* de desenvolvimento concerto F28M36x. Esse programa compreende um conjunto de ferramentas usadas para desenvolver aplicações embarcadas. É utilizado como ambiente de programação para esse projeto, sendo executado sobre um sistema operacional em tempo real (TI-RTOS).

3.2.3 TI-RTOS (*Texas Instruments Real Time Operational System*)

A *Texas Instruments* é a fabricante do *kit* de desenvolvimento concerto F28M36x e a mesma disponibiliza o sistema operacional TI-RTOS para utilização junto do *kit*. Este sistema operacional permite um desenvolvimento mais rápido das aplicações embarcadas, pois elimina a necessidade dos desenvolvedores de programar *drivers*. Possuindo componentes adicionais de *software*, como camadas TCP/IP e USB, sistemas de arquivos FAT e *drivers* de dispositivos. Além disso, o TI-RTOS fornece exemplos que demonstram como usar cada dispositivo suportado e drivers (TEXAS INSTRUMENTS, 2012).

3.2.4 Roteador D-Link Dl-524

É um roteador *Wireless* que atende ao padrão de conexão 802.11b e também oferece suporte ao padrão 802.11g. Em relação à velocidade e taxa de transmissão de dados, o roteador D-Link DI-524 é capaz de operar em até 150 Mbps com frequência de 2,4 GHz. Na Figura 12 é mostrado o roteador.

Figura 12 – Roteador D-link Dl 524.
Fonte: (COMUNIDADE HARDWARE, 2009)

3.2.5 Elipse SCADA

As características e funcionalidades do Elipse SCADA para a construção de sistemas de supervisão foram essenciais para a escolha do mesmo. O *software* fornece requisitos como telas configuráveis, *tags* de diferentes tipos, *displays*, botões e gráficos customizados, *driver* de comunicação com banco de dados ODBC (*Open Data Base Connectivity*) e comunicação com protocolo OPC.

3.3 Aquisição de Dados do Sistema de Micro Geração de Energia Hidráulica

Para aquisição de medidas do sistema de micro geração de energia hidráulica são utilizadas uma placa de interface com o DSP, uma placa de adaptação para conectar o DSP concerto F28M36x na placa de interface, placas para medições de tensões, placa para medição de corrente e placas para condicionamento das medidas às entradas analógicas do DSP. Um esquemático das placas utilizadas é mostrado na Figura 13.

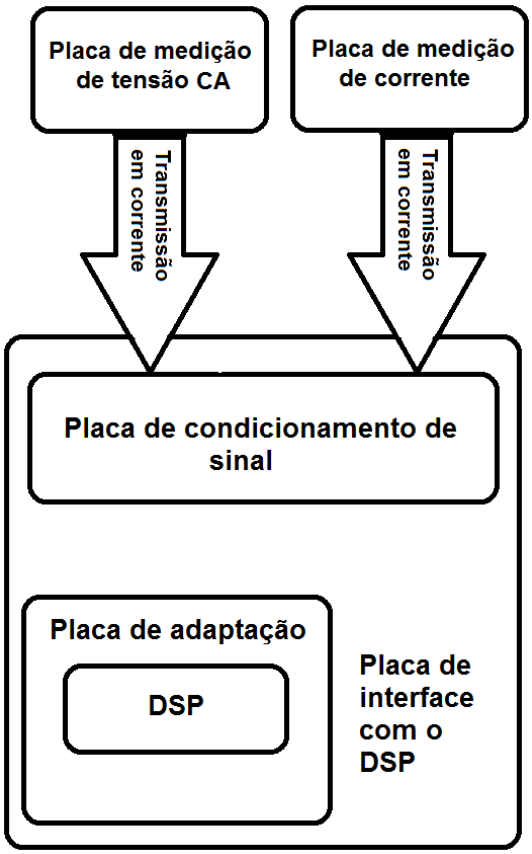


Figura 13 – Esquemático das placas de aquisição de medidas.

A placa de tensão possibilita a medida de duas tensões CA através de transformadores isolados. A placa de corrente contém três transdutores de efeito hall para medida de três correntes. As placas de condicionamento de sinal são utilizadas para adequar o sinal oriundo das placas de medição às entradas dos conversores A/D do DSP. Os sinais de medida provindos das placas de tensão e corrente são recebidos através de conectores do tipo DB9 na placa de interface.

A placa de interface com o DSP gerencia a comunicação entre as demais placas e o DSP, além de gerar as referências de tensão da instrumentação. Como a placa de interface foi criada para ser utilizada com o DSP TMS320F28335, foi desenvolvida uma nova placa de adaptação para o DSP concerto F28M36x conectar-se com a placa de interface. Assim, as placas de condicionamento de sinal e a placa de adaptação estão diretamente fixadas na placa de interface, e o DSP concerto F28M36x está diretamente fixado na placa de adaptação.

3.3.1 Placa de adaptação do DSP concerto F28M36x na placa de interface

Como explicado anteriormente, foi necessário desenvolver uma placa de adaptação para que o DSP concerto F28M36x conseguisse conectar-se e comunicar-se corretamente com a placa de interface.

Para o desenvolvimento da placa de adaptação foi utilizado o *software Altium Designer*. A placa consiste em adaptar os pinos da placa de interface e suas funcionalidades aos pinos do DSP concerto. A vista de cima (*top*) da placa pode ser vista na Figura 14.

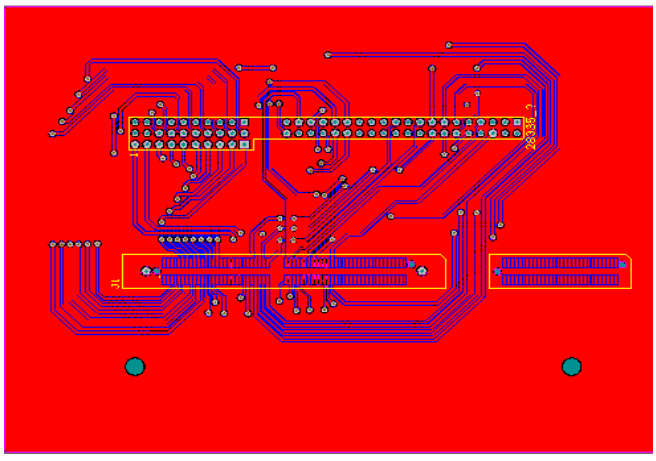


Figura 14 – Vista *top* da placa de adaptação.

3.4 Aquisição de Dados Pelo DSP Concerto F28M36x

O DSP concerto F28M36x realiza a aquisição dos dados que estão sendo gerados. Para isso, foi desenvolvida uma aplicação no núcleo C28x do DSP utilizando o *software* Code Composer, onde foi programado para o mesmo adquirir sinais da entrada analógica em tempo real. A aplicação desenvolvida se encontra no Apêndice A.

O código desenvolvido para a aplicação primeiramente precisa inicializar algumas tarefas, como: inicialização do controle dos subsistemas analógicos, habilitação de GPIO (*General Purpose Input/Output*) e inicialização dos *buffers* para alocação de variáveis. Posteriormente, é realizada a leitura das entradas analógicas do DSP.

Para que a aquisição de medidas em tempo real ocorra com sucesso é necessário configurar os módulos de conversão analógico-digital (ADC) e de modulação por largura de pulso (PWM). Nessa aplicação a conversão analógico-digital é disparada pelo módulo PWM à uma frequência de 10 kHz. Para cálculo da frequência de PWM é utilizada (1), mostrada abaixo (TEXAS INSTRUMENTS, 2009):

$$F_{PWM} = \frac{1}{2 * TBPRD * T_{Clock}} \tag{1}$$

onde,

F_{PWM} = frequência do PWM,

$TBPRD$= registrador do período de base de tempo,

T_{Clock}= *clock* de base de tempo, inverso da frequência de operação da CPU (150 MHz).

assim, é feito o cálculo para F_{PWM}= 10 kHz.

$$10kHz = \frac{1}{2 * TBPRD * (1/150\ MHz)}$$

$$TBPRD = 7500$$

Com a frequência de operação do módulo PWM e o valor do registrador TBPRD definidos, é necessário realizar a parametrização do módulo PWM1 do DSP, que fará o disparo da interrupção da conversão analógico-digital ADC1 e ADC2.

3.4.1 Cálculos realizados a partir dos dados adquiridos

Os dados adquiridos pelo DSP foram as correntes de carga I_a, I_b, I_c e as tensões de linha V_{ab} e V_{bc}. Foi necessário calcular o *offset* e o ganho de cada medida, para depois transformar as tensões de linha para fase, conforme (2), (3) e (4) (SINGH, 2014).

$$V_{a\,fase} = \frac{1}{3} * ((2 * V_{ab}) + V_{bc}) \tag{2}$$

$$V_{b\,fase} = \frac{1}{3} * (-V_{ab} + V_{bc}) \tag{3}$$

$$V_{c\,fase} = (-V_{a\,fase} - V_{b\,fase}) \tag{4}$$

Com isso, foi possível calcular a potência ativa instantânea, potência reativa instantânea, potência aparente e fator de potência conforme (5), (6), (7) e (8) respectivamente.

$$P_{ativa} = \left(V_{a\,fase} * I_{a\,carga}\right) + \left(V_{b\,fase} * I_{b\,carga}\right) + \left(V_{c\,fase} * I_{c\,carga}\right) \tag{5}$$

$$P_{reativa} = \left(\left((V_{a\,fase} - V_{b\,fase}) * I_{c\,carga}\right) + \left((V_{b\,fase} - V_{c\,fase}) * I_{a\,carga}\right) + \left((V_{c\,fase} - V_{a\,fase}) * I_{c\,carga}\right)\right) / \sqrt{3} \tag{6}$$

$$P_{aparente} = \sqrt{{P_{ativa}}^2 + {P_{reativa}}^2} \tag{7}$$

$$FP = P_{ativa}/P_{aparente} \tag{8}$$

Devido à presença de ruídos na medição foi utilizado o método de média móvel para filtragem do sinal. A média móvel é um dos filtros mais comuns utilizados em processamento digital de sinais, principalmente, porque é um dos filtros digitais mais fáceis de entender e utilizar. Além da simplicidade, o filtro de média móvel é ótimo para reduzir ruído randômico enquanto mantém uma rápida resposta ao degrau. Isto faz com que a média móvel seja um excelente filtro para sinais codificados no domínio do tempo (BOTTERÓN, ANDRADE, 2002).

O filtro de média móvel é obtido calculando-se a média de um conjunto de valores, sempre adicionando um novo valor ao conjunto e descartando um antigo. Não é apenas uma média de um conjunto isolado de valores. O filtro pode ser representado por (9), abaixo:

$$y[n] = \frac{1}{N+1}\sum_{k=0}^{N} x[n-k] \quad (9)$$

onde,

$n =$ é o índice dos vetores utilizados;

$N + 1 =$ é o número de amostras utilizadas para a filtragem;

$y[n] =$ é o sinal filtrado;

$x[n-k] =$ é o conjunto de valores a serem somados.

3.5 Implementação da Comunicação *Modbus* TCP/IP no DSP Concerto F28M36x

Para realizar a comunicação *Modbus*, o núcleo Cortex-M3 e o núcleo C28x do DSP foram programados utilizando o *software* Code Composer. O núcleo Cortex-M3 é responsável pela comunicação *Modbus* TCP/IP com o sistema de supervisão, já o núcleo C28x é responsável pela aquisição de sinais a partir da entrada analógica do DSP. Como a comunicação é realizada pelo padrão TCP/IP, o DSP precisa ser configurado para esta função. Através do TI-RTOS é possível configurar a comunicação TCP/IP e selecionar os *drivers* de EMAC (*driver* de *ethernet* usado pelo protocolo de rede), GPIO e UART (receptor/transmissor de dados assíncrono universal), como mostrado na Figura 15.

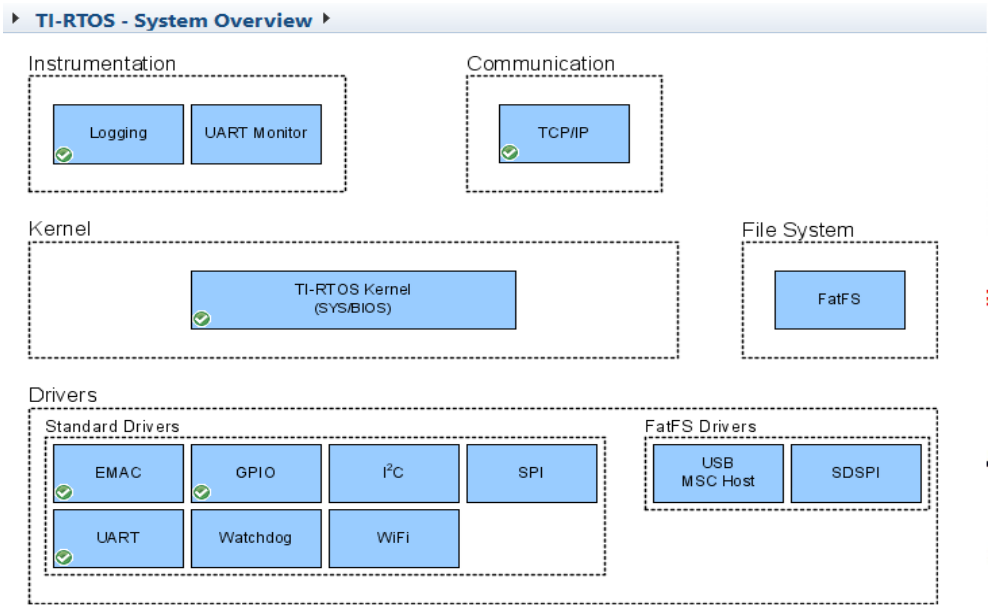


Figura 15 – Configuração TI-RTOS.

Realizada a configuração do sistema operacional com os *drivers* necessários, foram desenvolvidas aplicações nos núcleos Cortex-M3 e C28x com configuração de servidor para a comunicação *Modbus* TCP/IP, essas aplicações se encontram no Apêndice A e Apêndice B.

Como o núcleo C28x é responsável pela aquisição de sinais da entrada analógica e o núcleo Cortex-M3 pela comunicação *Modbus* TCP/IP, é necessário realizar a comunicação entre esses processadores. Então nas aplicações realizadas é utilizado um módulo MessageQ, este módulo suporta uma estrutura de envio e recebimento de mensagens de tamanho variável. A utilização do MessageQ é o que permite o núcleo C28x enviar as variáveis analógicas adquiridas para o núcleo Cortex-M3, que posteriormente transmitirá as variáveis via *Modbus* TCP/IP.

O código desenvolvido para a aplicação no núcleo Cortex-M3 necessita inicializar algumas tarefas para realizar a comunicação entre os núcleos do DSP, como: inicialização dos *buffers* para alocação de variáveis e habilitação da estrutura MessageQ. Esse núcleo possui duas funções importantes para a realização da comunicação TCP/IP, as funções *tcpHandler* e *tcpWorker*.

A função *tcpWorker* realiza o envio e recebimento de dados quando a conexão está estabelecida. Através dessa função ocorre o recebimento da requisição do cliente e também realiza a verificação se a mensagem de requisição é para o endereço de escravo (servidor) do DSP, conforme Figura 16. Após a verificação é possível enviar a resposta ao cliente. Essa resposta possui o mesmo endereço de escravo (servidor), a mesma identificação de transação e o mesmo código de função da requisição, juntamente com os dados requisitados. Nesse código foram implementados três códigos de funções: leitura de registradores (função 3), escrita em um registrador (função 6) e escrita em vários registradores (função 16). A Figura 17, Figura 18 e Figura 19 apresentam os códigos das funções implementadas.

```
// ----- MENSAGEM PARA MIM -------------------------------------
 if (BilheteModbusR[6] == EnderecoModbus) // Se mensagem é para este endereço (primeiro byte)
 {    auxva=22;

    BilheteModbusT[6] = BilheteModbusR[6]; // Endereço do bilhete a ser enviado = recebido
    BilheteModbusT[7] = BilheteModbusR[7]; // Código Função do bilhete a ser enviado = recebido
            if (nbytes3 < 256) {
    BilheteModbusT[1] = BilheteModbusR[1]; // Identificação de transação a ser enviado = recebido
    BilheteModbusT[0] = 0;

                              }
            if (nbytes2 > 0) {               //Cado o identificador de transação seja de valor maior que 255
              BilheteModbusT[0] = (nbytes2);  //nbytes2 = (BilheteModbusR[0])
              BilheteModbusT[1]= (nbytes3);   //nbytes3 = (BilheteModbusR[1])
                              }
```

Figura 16 – Código de verificação da requisição.

```
//----- FUNÇÃO 03 = LEITURA --------------------------------
case 3: // Se BilheteModbusR [7] = 3
   NumDados = ((BilheteModbusR[10]<<8 & 0xFF00) + (BilheteModbusR[11])*2); // Número de bytes a serem lidos
   iT = 8;
   auxva=10;

   for (i=EndInic; i < (EndInic + (NumDados)); i++)
   {
      iT++;
      auxva=12;
      BilheteModbusT[iT] = BufferModbus[i];
   }

   BilheteModbusT[8] = NumDados;    // O número de dados (byte alto + byte baixo = 1 dado)
   BilheteModbusT[5] = (NumDados)+3;    // O número de dados (byte alto + byte baixo = 1 dado)
```

Figura 17 – Código da função de leitura (função 03).

```
//----- FUNÇÃO 06 = ESCREVE 1 REGISTRADOR -------------------
    case 06:
       BufferModbus[EndInic] = BilheteModbusR[10];      // Escreve parte alta
       BufferModbus[EndInic + 1] = BilheteModbusR[11]; // Escreve parte baixa
       auxva=50;
       for (iT=8; iT<= 11; iT++)
       {
          BilheteModbusT[iT] = BilheteModbusR[iT];  // O bilhete a ser transmitido é igual ao recebido
       }
       iT = 11;
        break;
```

Figura 18 – Código da função de escrita em um registrador (função 06).

```
//----- FUNÇÃO 16 = ESCREVE VÁRIOS REGISTRADORES ------------
     case 16:
          NumDados = ((BilheteModbusR[10]<<8 & 0xFF00) + (BilheteModbusR[11])*2); // Número de bytes a serem escritos

          iT = BilheteModbusR[12]; // Número de bytes de dados

          iT = iT + 12;
         BilheteModbusT[5] = (NumDados)+4;    // O número de dados (byte alto + byte baixo = 1 dado)

          for (i=13; i<=iT; i++)
          {
             BufferModbus[EndInic] = BilheteModbusR[i]; // Escreve no Buffer do DSP os dados do Bilhete
             EndInic++;
          }

          for (i=8; i<= 14; i++)
          {
             BilheteModbusT[i] = BilheteModbusR[i];  // O bilhete a ser transmitido é igual ao recebido
          }                                          // até o byte 6
          iT = 14;
         break;
```

Figura 19 – Código da função de escrita em vários registradores (função 16).

A função *tcpHandler* inicia e cria conexões TCP através de um *socket* (interface entre a camada de aplicação e de transporte), assim consegue realizar as conexões.

3.6 Configuração do Sistema de Supervisão para a Comunicação *Modbus* TCP/IP

Foi desenvolvido no computador um sistema de supervisão SCADA através do *software* Elipse SCADA, em que o sistema de supervisão (cliente) efetua a leitura e atualização dos dados transmitidos pelo DSP. Para tanto foi parametrizado o *driver Modbus* no Elipse SCADA e configurada as variáveis para que o cliente (sistema de supervisão) realize a comunicação com o servidor (DSP). Esse *driver* implementa o protocolo *Modbus*, permitindo ao Elipse atuar como cliente ou mestre em uma rede *Modbus*, comunicando-se com qualquer equipamento escravo ou servidor que implemente os protocolos *Modbus*. Primeiramente é necessário fazer o *download* do *driver* no site do Elipse SCADA, assim é possível carregar o *driver* no supervisório como mostrado na Figura 20.

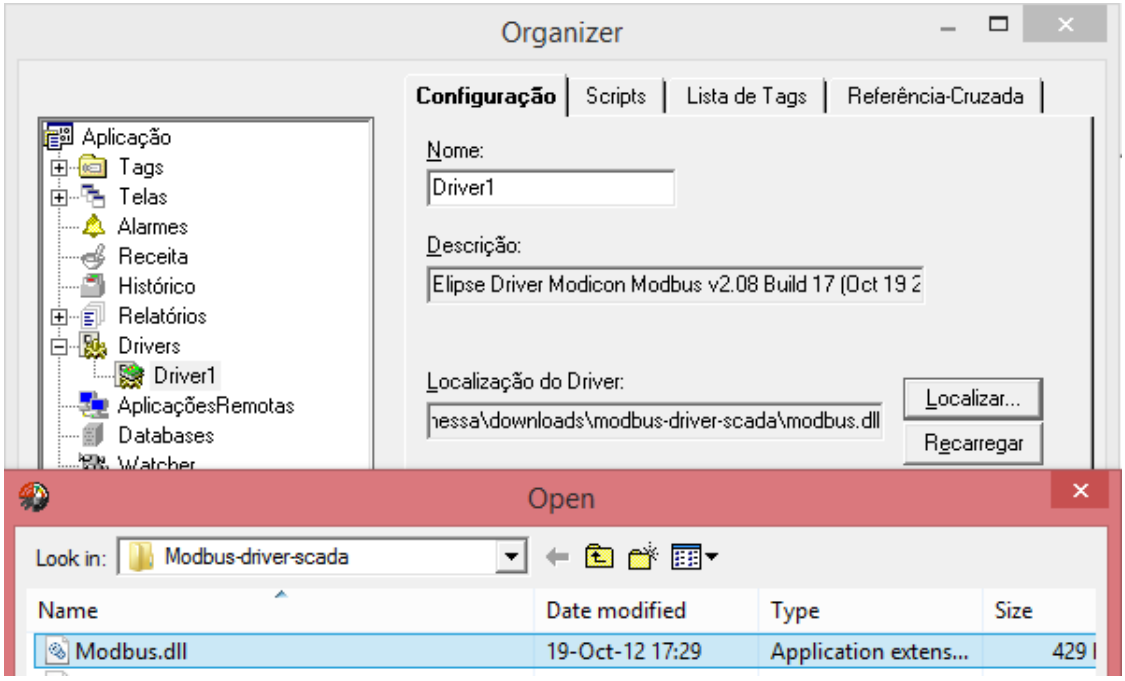


Figura 20 – Localizando o *driver* e utilizando no sistema de supervisão.

Conforme mostra a Figura 21 o *driver* foi configurado para comunicação *Modbus* TCP/IP com o endereço do servidor (escravo) sendo de valor 1. Na Figura 22 é selecionada a camada física *Ethernet*, através dessa camada ocorrerá a transmissão de dados. Já na Figura 23 é mostrado a escolha da camada de transporte (TCP/IP), o endereço de IP do DSP e a porta escolhida para comunicação.

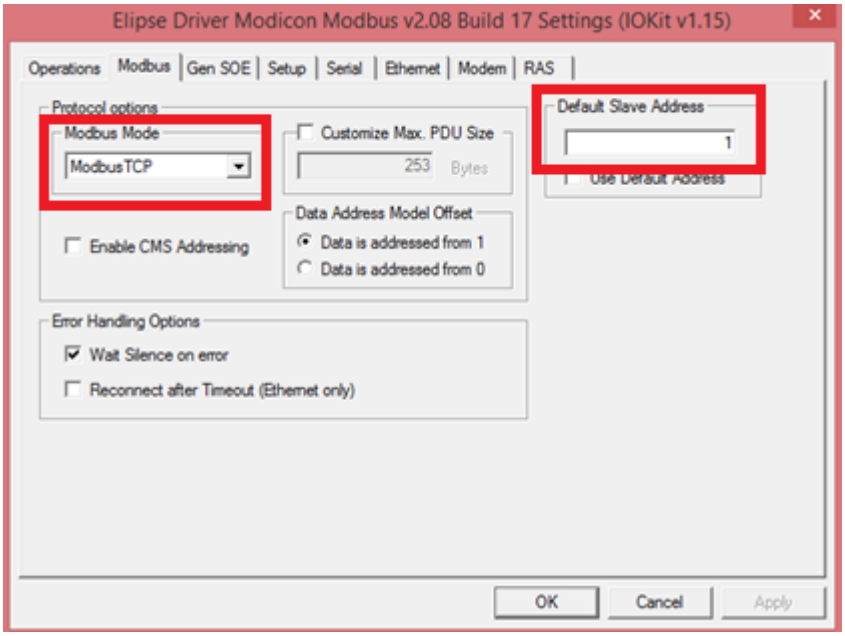


Figura 21 – Configuração do *driver Modbus* TCP/IP.

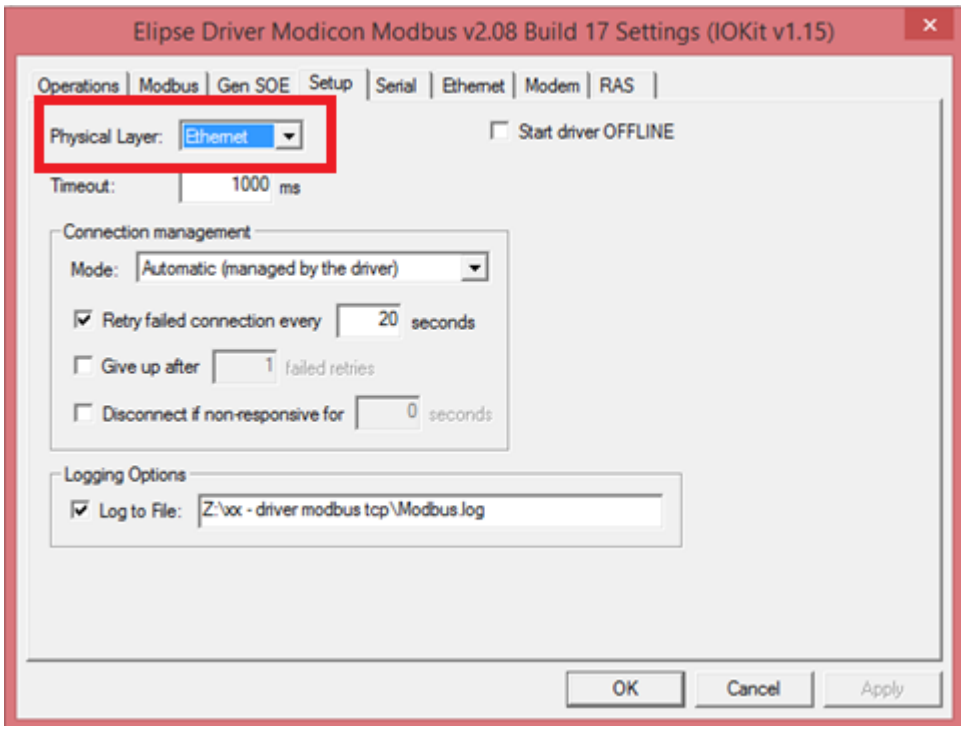


Figura 22 – Configuração da camada física.

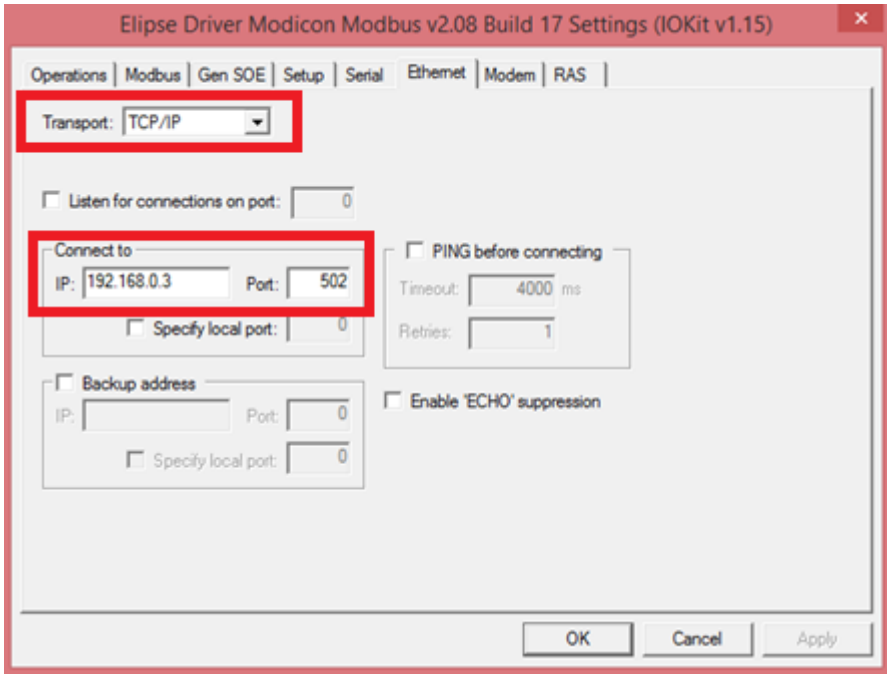


Figura 23 – Configuração Ethernet do *driver*.

Para leitura das variáveis no supervisório é necessário a parametrização das variáveis. No *software* utilizado as variáveis são reconhecidas por *tags*. Para cada *tag* é necessário definir valores de N1 (endereço do equipamento escravo na rede), N2 (código da operação), N3 (parâmetro adicional) e N4 (endereço do registrador ou da variável do equipamento escravo que deseja ler ou escrever). Na Figura 24 é possível observar a parametrização de uma *tag*.

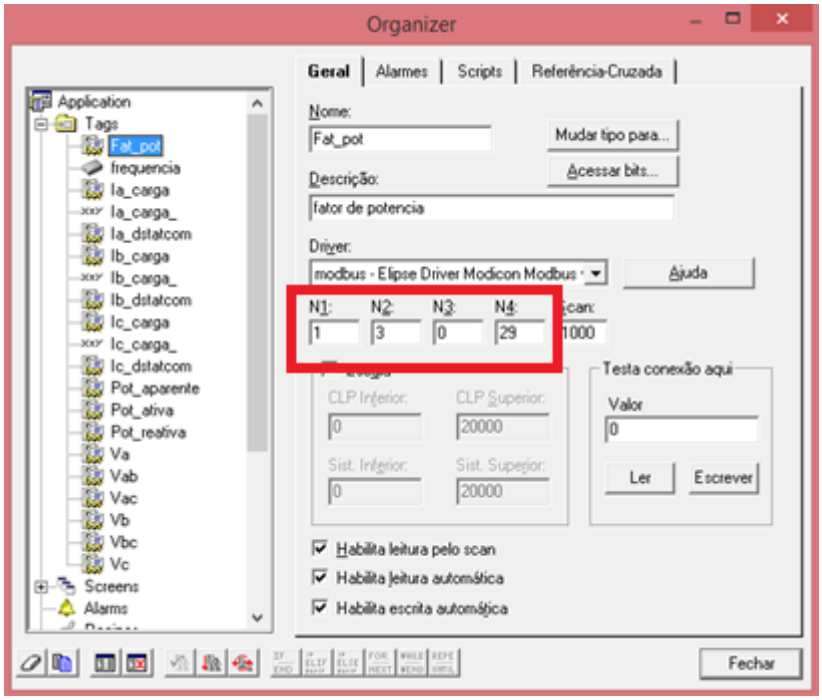


Figura 24 – Parametrização de *tags*.

3.7 Desenvolvimento do Sistema de Supervisão

O sistema de supervisão foi desenvolvido no computador utilizando o software Elipse SCADA. O sistema supervisório efetua a leitura e atualização dos dados transmitidos pelo DSP em tempo real através do protocolo *Modbus* TCP/IP. O supervisório desenvolvido possui várias telas e funcionalidades, tendo a intenção de possuir uma interface amigável para fácil operação e visualização dos dados adquiridos do sistema de micro geração de energia hidráulica.

3.7.1 Tela de *Login*

Primeiramente foi criado um sistema básico de *login* de usuário. Para isso é necessário criar o usuário que terá acesso às telas de supervisão. Com a criação do usuário realizada, é necessário configurar um botão com a mensagem "Login" que terá um *script* no evento *OnPress* (ao clicar). O *script* contém o comando "Aplicação.Login()", que mostrará um campo para inserção de usuário e senha, que já foram previamente definidos. Ao pressionar "Ok", caso os dados estejam corretos, a aplicação vai entender que há um usuário inserido no sistema. A aplicação foi programada para que, caso o usuário e senha estejam corretos, o usuário seja direcionado para a página principal do sistema de supervisão. Caso o usuário ou senha estejam incorretos, a aplicação retornará uma mensagem de erro. A tela de *login* é apresentada na Figura 25.

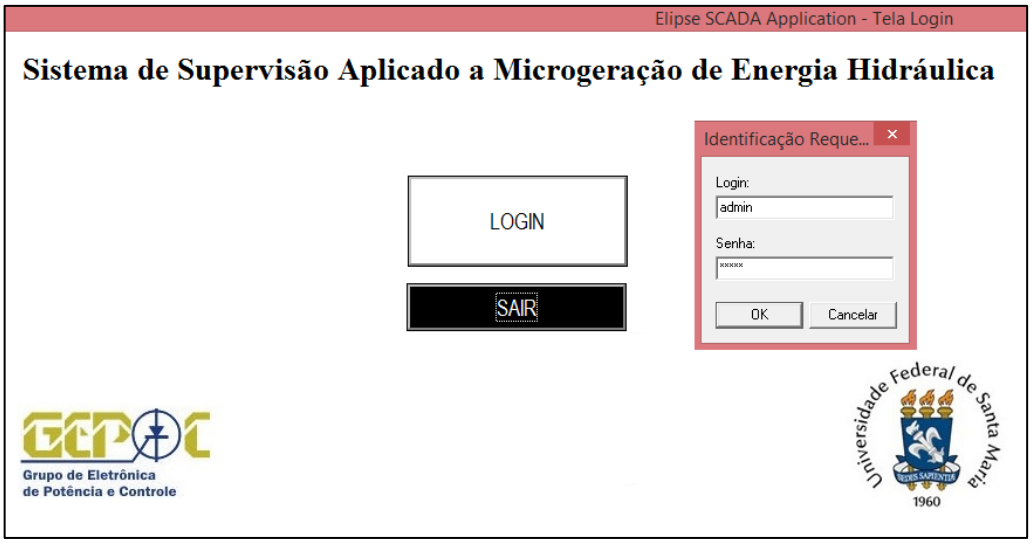


Figura 25 – Tela de *login* com campo de preenchimento de usuário.

3.7.2 Tela Principal

A tela principal do sistema de supervisão encontra-se na Figura 26. A tela principal possui uma série de botões que direcionam o usuário a outras telas que possuem diferentes informações do sistema supervisionado. Possui botões para o usuário fazer o *logout* e também para fechar a aplicação.

Nesta tela é possível encontrar informações em tempo real sobre o *status* de funcionamento do gerador, a potência ativa instantânea, potência reativa instantânea, potência aparente instantânea, fator de potência e frequência. Todas essas informações serão mostradas nos quadros em branco presentes na tela e estão relacionadas a *tags* configuradas para adquirirem dados do DSP.

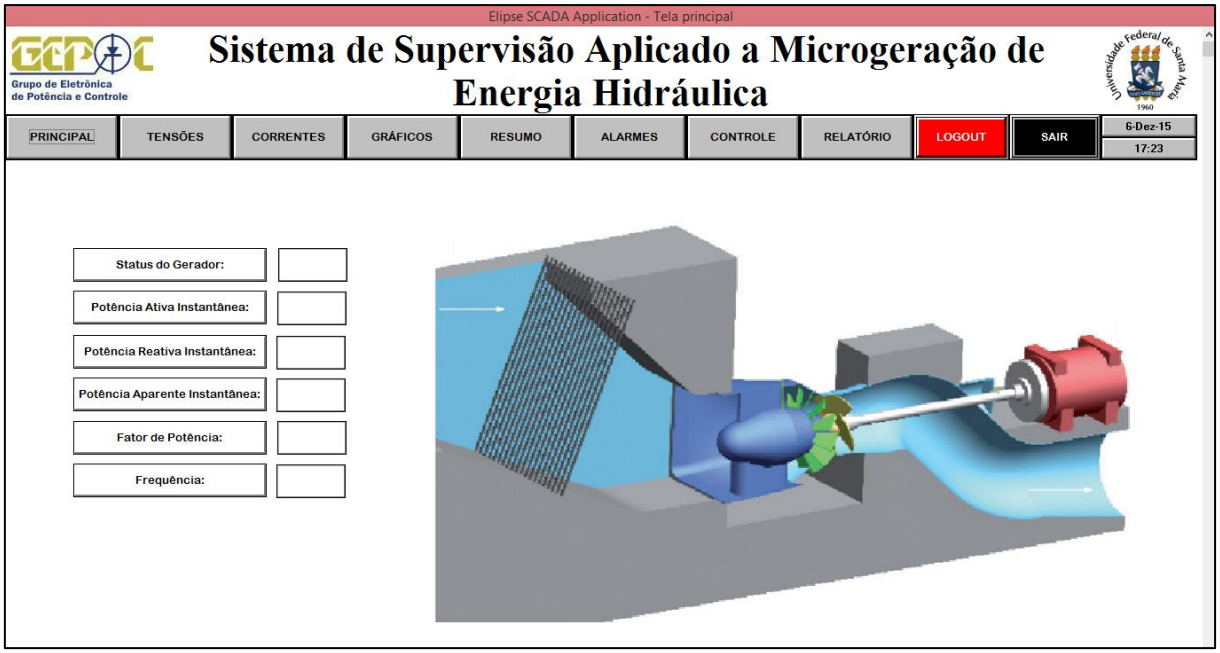


Figura 26 – Tela principal do sistema de supervisão.

3.7.3 Tela de Tensões

Ao clicar no botão “Tensões”, o usuário será direcionado a tela de medições de tensão, conforme Figura 27.

Na tela de tensões é possível visualizar dados em tempo real sobre tensão de linha V_{ab}, V_{bc}, V_{ac}, tensão de fase V_a, V_b e V_c do gerador.

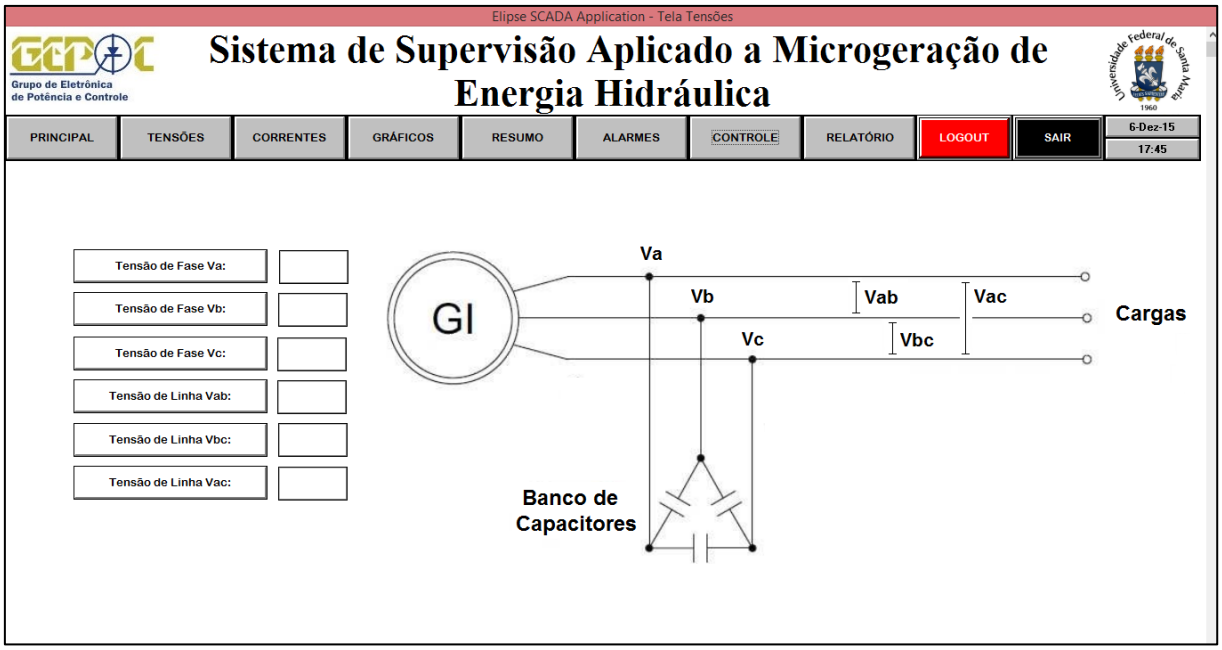


Figura 27 – Tela de tensões do sistema de supervisão.

3.7.4 Tela de Correntes

Ao clicar no botão “Correntes”, o usuário será direcionado a tela de medições de corrente, conforme Figura 28.

Nesta tela de correntes é possível visualizar valores em tempo real de corrente de carga I_a, I_b, I_c, corrente do DSTATCOM I_{aD}, I_{bD}, I_{cD}, corrente do gerador I_{aG}, I_{bG} e I_{cG}.

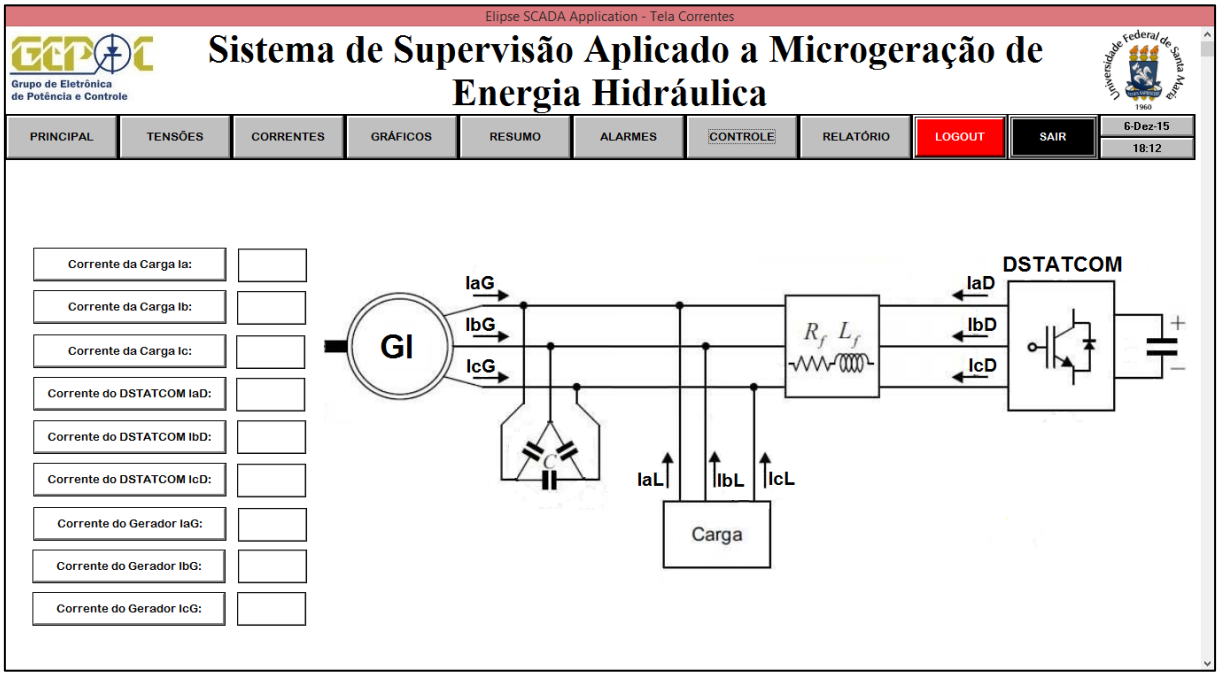


Figura 28 – Tela de correntes do sistema de supervisão.

3.7.5 Tela de Gráficos

A tela de gráficos é mostrada na Figura 29. Nesta tela é possível visualizar gráficos em tempo real de potências, tensões e correntes. Para selecionar que tipo de gráfico o usuário deseja visualizar basta ele clicar nos botões que especificam o gráfico que será gerado.

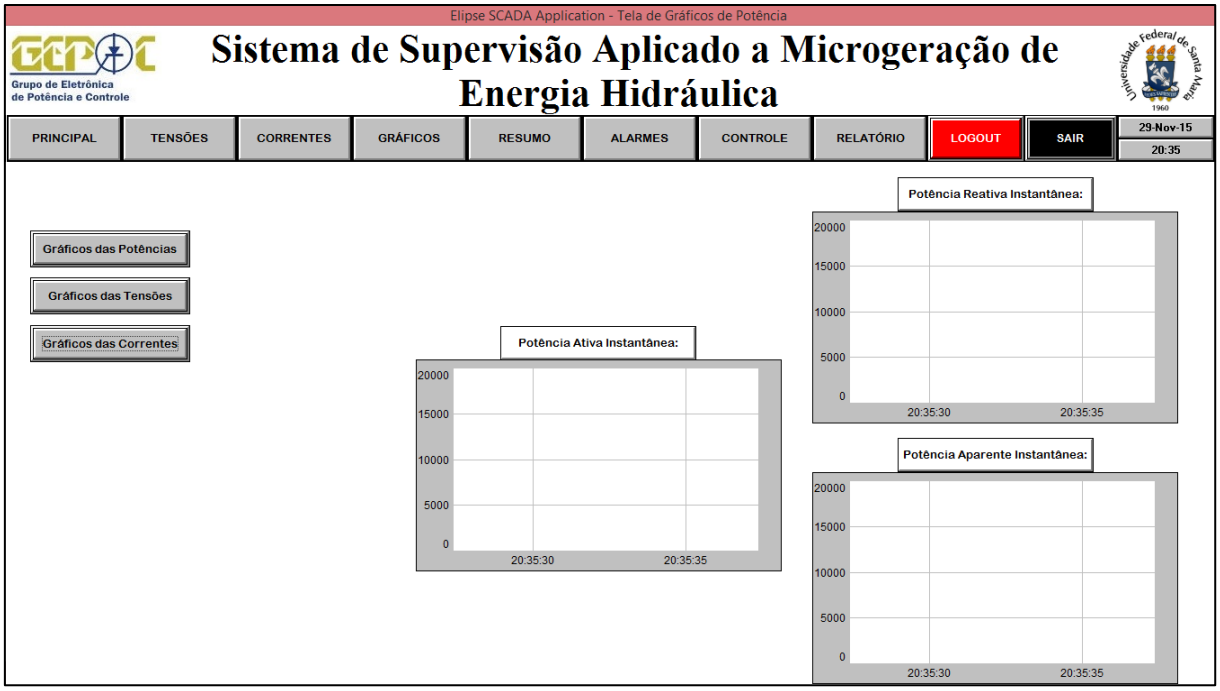


Figura 29 – Tela de gráficos do sistema de supervisão.

3.7.6 Tela de Resumo

A tela de resumo é uma tela que contém todos os dados recebidos do DSP, como apresenta a Figura 30. Nesta tela é possível visualizar em tempo real informações como: status do gerador, potência ativa instantânea, potência reativa instantânea, potência aparente instantânea, fator de potência, frequência, tensão de linha V_{ab}, V_{bc}, V_{ac}, tensão de fase V_a, V_b, V_c, corrente de carga I_a, I_b, I_c, corrente do DSTATCOM I_{aD}, I_{bD}, I_{cD}, corrente do gerador I_{aG}, I_{bG} e I_{cG}.

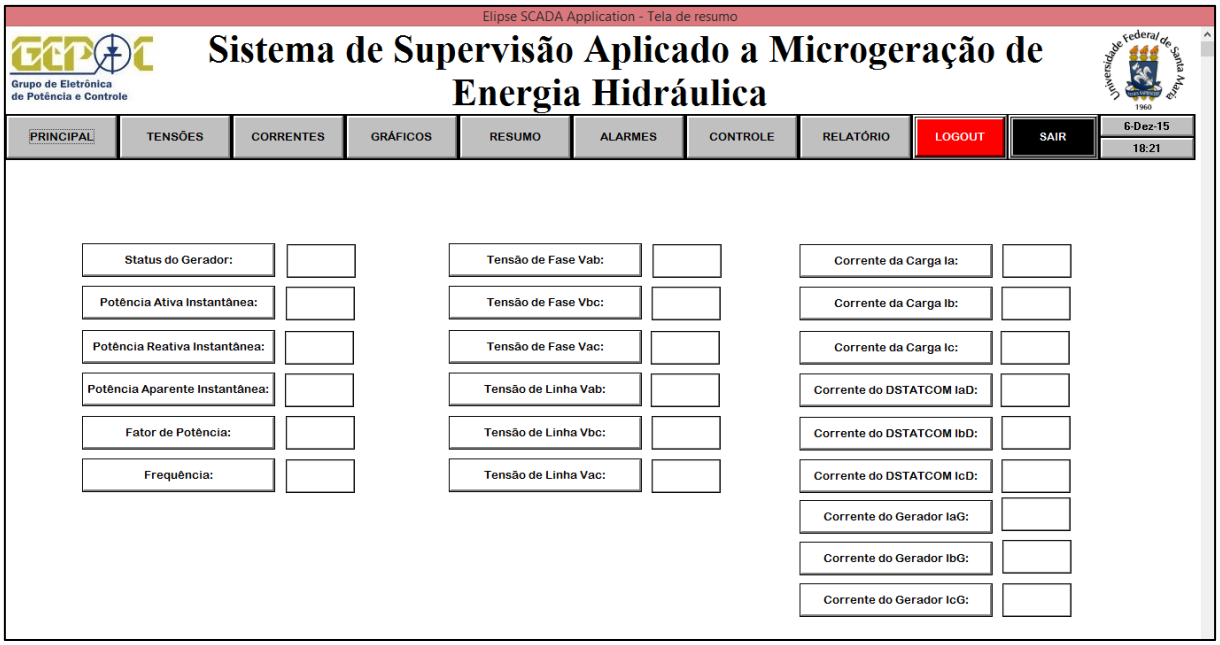


Figura 30 – Tela de resumo do sistema de supervisão.

3.7.7 Tela de Alarmes

A tela de alarmes mostrará o *status* de funcionamento atual do sistema. Caso a tensão de linha ultrapasse o valor de 400 V ou as correntes de cargas ultrapassem o valor de 3 A, serão mostrados os alarmes na janela de alarme e o *status* de funcionamento altera-se para "Em risco". Do contrário, o sistema permanecerá em estado de funcionamento normal. A tela pode ser vista na Figura 31.

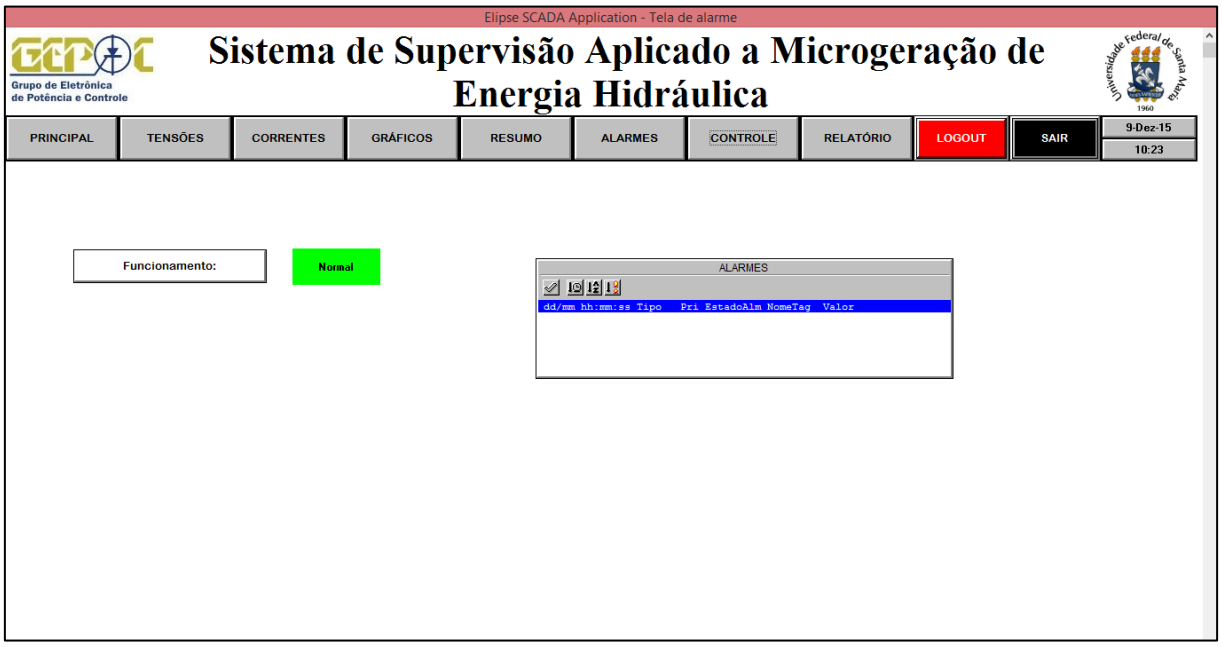


Figura 31 –Tela de alarmes do sistema de supervisão

3.7.8 Tela de Controle

A tela de controle tem como funcionalidade alterar e visualizar parâmetros de controle do sistema, segundo mostrado na Figura 32. Ao clicar no bloco de “Controladores de tensão CA e CC”, será mostrado uma nova janela com os parâmetros do controlador, onde os mesmo são exibidos e podem ser alterados pelo usuário. O usuário terá informações do *status* do gerador, da potência ativa instantânea e da potência reativa instantânea, assim ao alterar os parâmetros de controle é possível analisar como o sistema comporta-se.

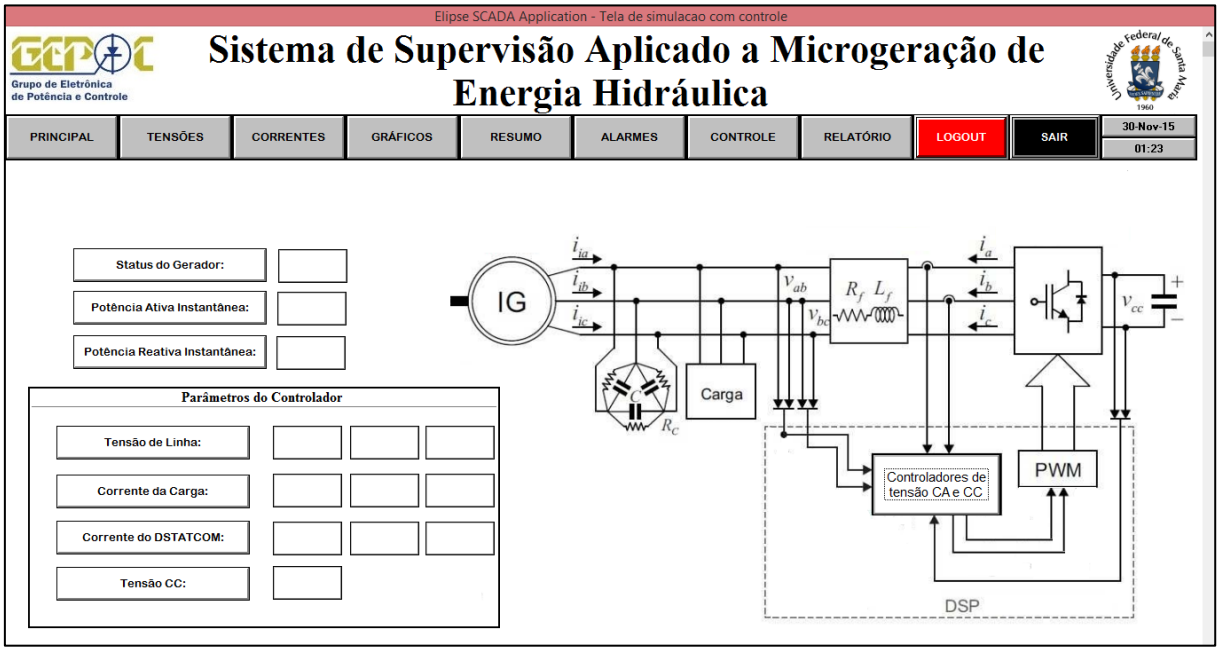


Figura 32 – Tela de controle do sistema de supervisão.

Como neste trabalho não foi desenvolvido o controle do sistema de micro geração hidráulica e aplicado no DSP, foi utilizado um modelo desse sistema que possui controladores de tensão CA e CC no ambiente de software *Simulink*/*Matlab*.

Ao utilizar o *Simulink* é possível realizar a comunicação do mesmo com o sistema de supervisão através do protocolo OPC. Para essa comunicação ser efetuada é preciso, primeiramente, criar um servidor OPC na aplicação do Elipse SCADA, conforme Figura 33.

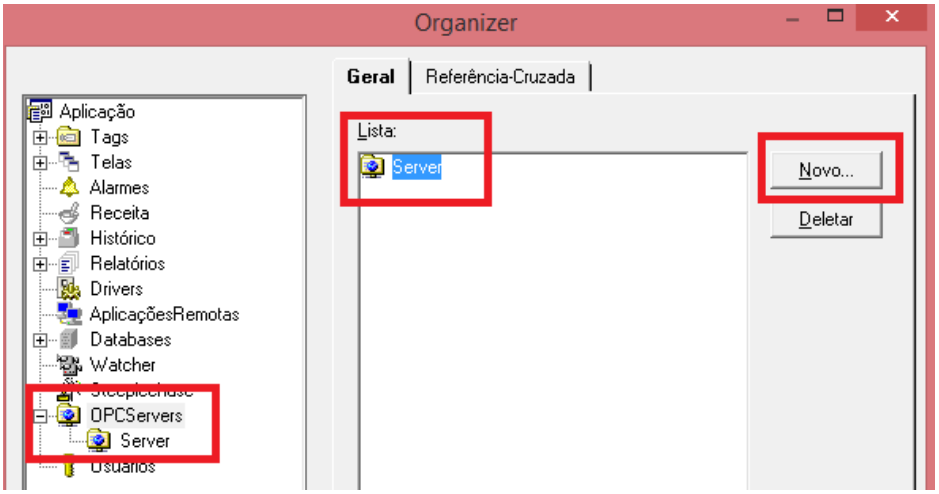


Figura 33 – Criação de servidor OPC.

Após criar o servidor, é necessário configurar o mesmo. Seleciona-se o endereço na rede, neste caso o *localhost*, e o ID do servidor, neste caso é o "ElipseSCADA.OPCSvr.1". A Figura 34 mostra a janela de configurações.

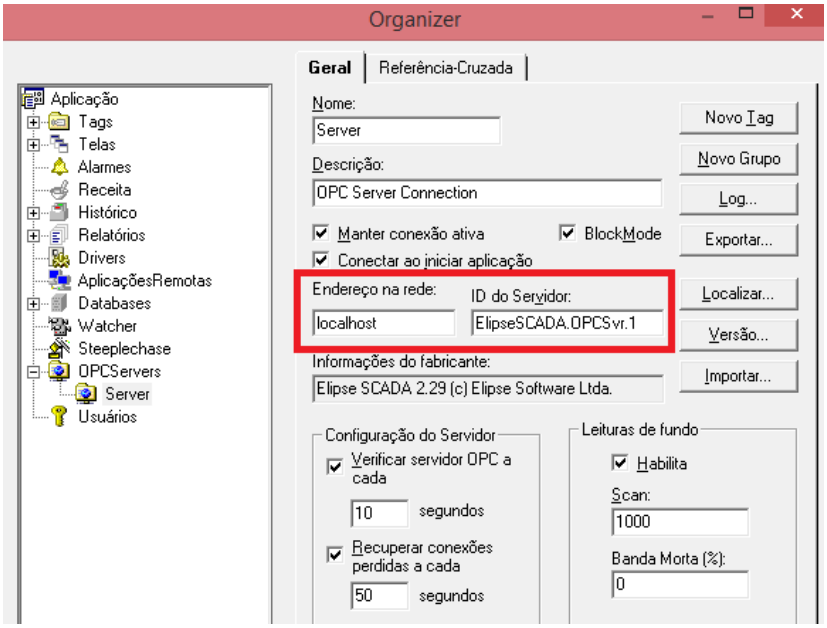


Figura 34 – Configurações do servidor OPC.

Para configurar a comunicação OPC no *Simulink*, deve-se adicionar os blocos de configuração OPC no ambiente de simulação e configurá-lo, conforme mostrado na Figura 35. O campo *host* precisa ser igual o endereço de rede escolhido no Elipse SCADA para realizar-se a comunicação, então utiliza-se o *localhost.* Da mesma maneira, é escolhido "ElipseSCADA.OPCSvr.1" como *server*.

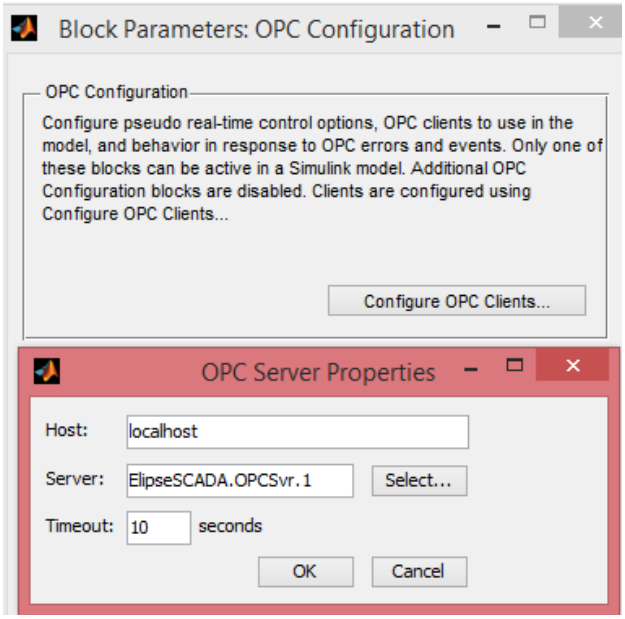


Figura 35 – Configuração da comunicação OPC no *Simulink*.

Além disso, também é necessário adicionar blocos de leitura e escrita de parâmetros através do padrão OPC no ambiente de simulação, conforme mostrado na Figura 36. Esses blocos também precisam ser configurados, de maneira que, as variáveis que serão enviadas ou recebidas deverão estar relacionadas as *tags* do sistema supervisório.

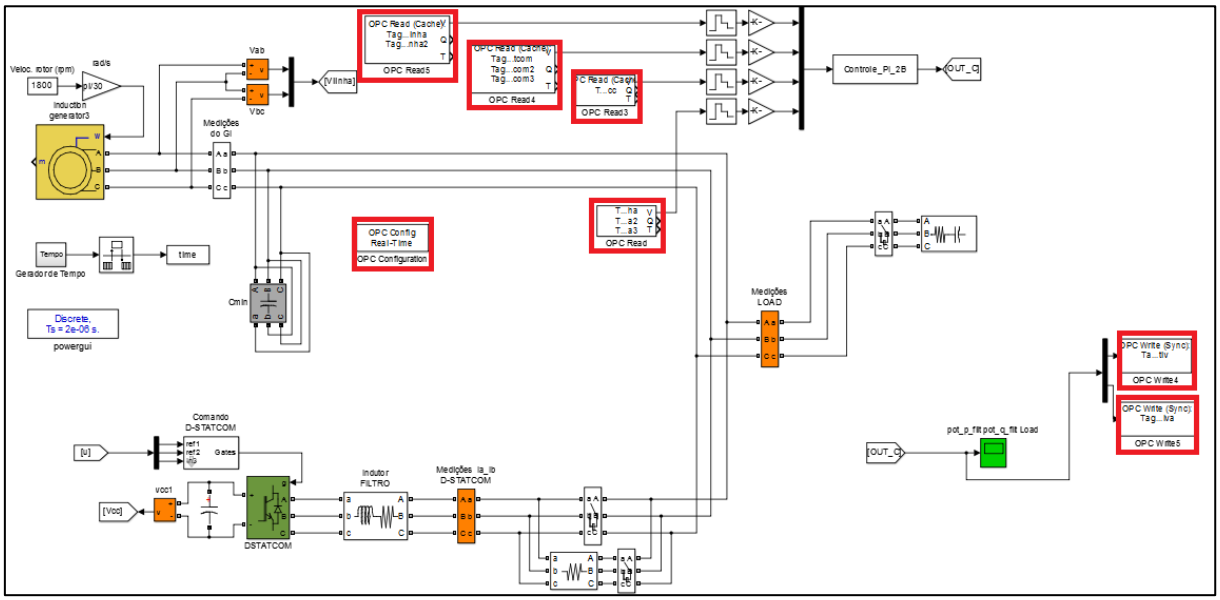


Figura 36 – Blocos OPC no ambiente *Simulink*.

3.7.9 Tela de Relatório

A tela de relatório permite ao usuário gerar um relatório das variáveis monitoradas. Esta tela permite o usuário gerar um relatório gráfico e também um relatório em formato de texto, como apresentado na Figura 37.

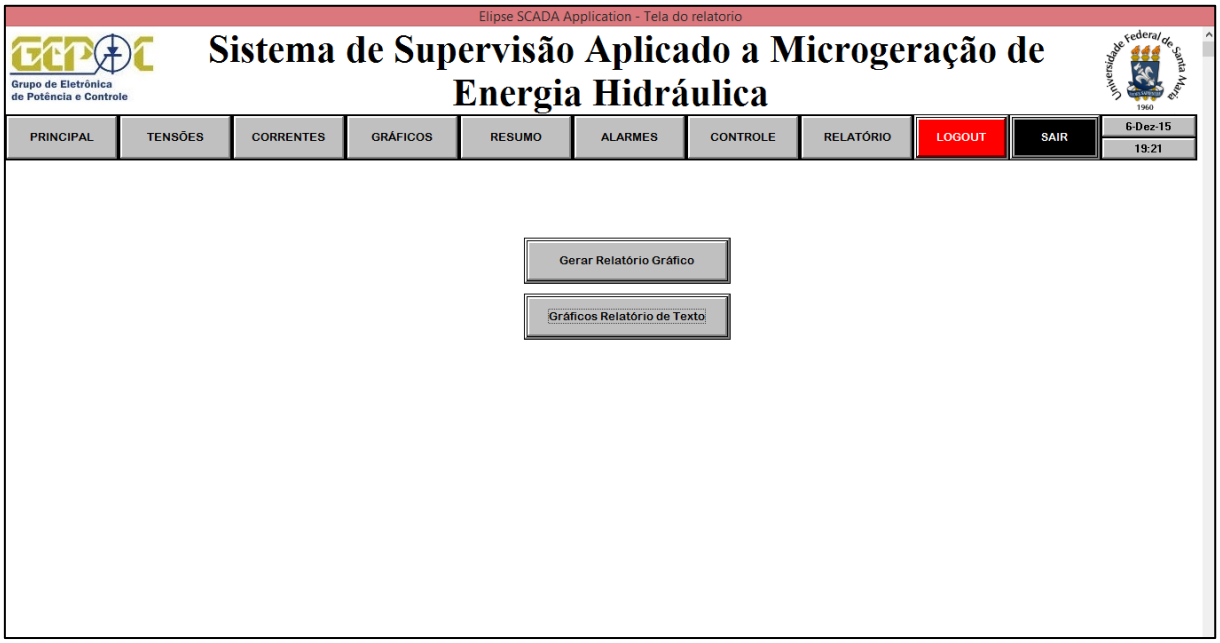


Figura 37 – Tela de relatório do sistema de supervisão.

Para a funcionalidade de gerar relatórios, primeiramente, é necessário que seja criado um histórico no sistema de supervisão. O histórico armazena os valores das *tags* selecionadas continuamente durante a execução da aplicação. Assim, o relatório será gerado baseado nos dados previamente configurados no histórico.

O relatório será gerado no momento que o usuário clica no botão referente ao tipo de relatório que deseja. Ao clicar no relatório texto, o botão possui um *script Onpress* (ao clicar), que utiliza o comando “Relatórios.Relatório1.PrintToFile("C:\arquivo.txt",1," ") para gerar o relatório e salvar no computador. No caso do relatório gráfico, o botão possui também um *script* que utiliza o comando “Relatório2.Print(0)”. No caso do relatório gráfico, o comando é para imprimir o relatório através de uma impressora, mas foi utilizado uma impressora PDF para gerar o relatório em formato PDF no computador, conforme Figura 38.

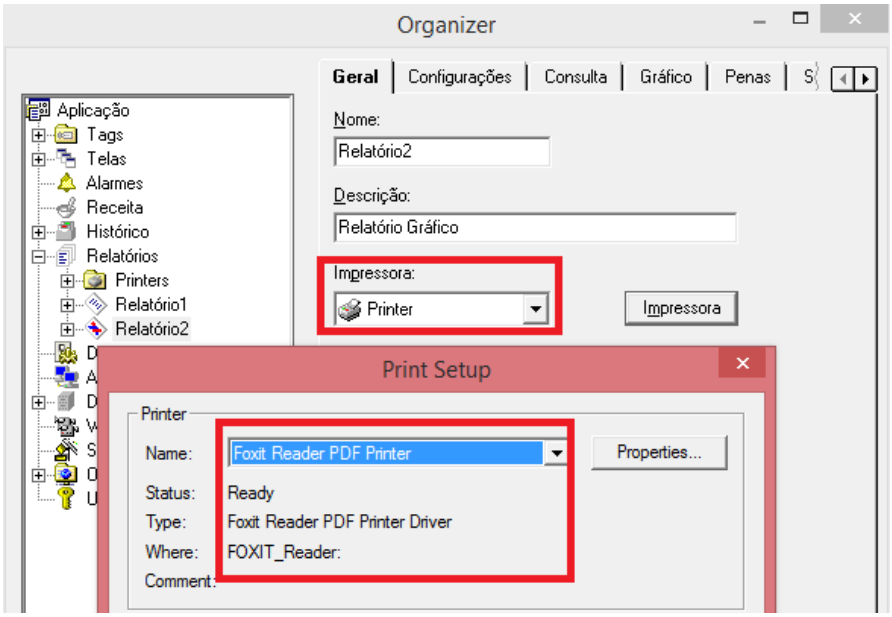


Figura 38 – Configurando relatório gráfico para impressora PDF.

3.8 Criação de um Banco de Dados para o Sistema de Supervisão

O *software* Elipse SCADA permite conectar e manipular um ou mais banco de dados utilizando o padrão ODBC (*Open Database Connectivity*). O ODBC é um padrão para acesso a sistemas gerenciadores de banco de dados. Para utilizar esse padrão é necessário ter um *driver* ODBC instalado no computador.

Para este projeto foi escolhido utilizar o MySQL como sistema de gerenciamento de banco de dados. O mesmo foi escolhido por ser um dos mais populares existentes, por suas características de rápido acesso, pela alta compatibilidade com a linguagem PHP e por ser otimizado para aplicações *Web* (INFOWESTER, 2008). É muito comum encontrar serviços de hospedagem de sites que oferecem o MySQL. Sendo assim, foi utilizado o banco de dados MySQL do site de hospedagem gratuita HelioHost.org.

Primeiramente, foi criado um banco de dados no servidor *Web* do site HelioHost.org. Assim, foi possível criar uma conexão ODBC, essa conexão é configurada com o *driver* ODBC a partir de informações do banco de dados já criado na *Web*, como apresentado na Figura 39. É escolhido um nome para a conexão realizada, selecionado o servidor, o usuário, a senha e o nome do banco de dados. É possível testar a comunicação entre o *driver* e o banco de dados para avaliar se a conexão foi efetuada com sucesso, sendo importante salientar que é necessária conexão com a *internet* para efetuar a conexão.

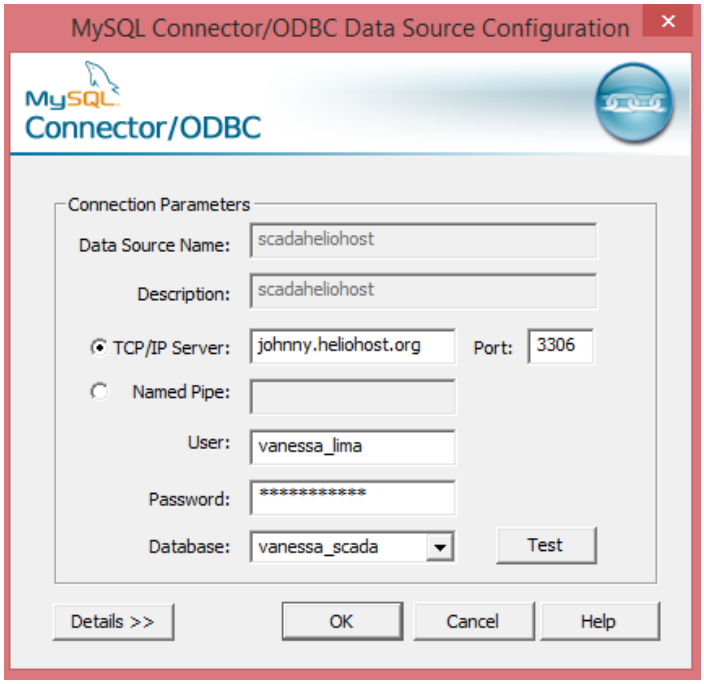


Figura 39 – Configuração do *driver* ODBC.

Finalmente, com o conector ODBC configurado, é possível acessá-lo pelo Elipse SCADA, conforme Figura 40.

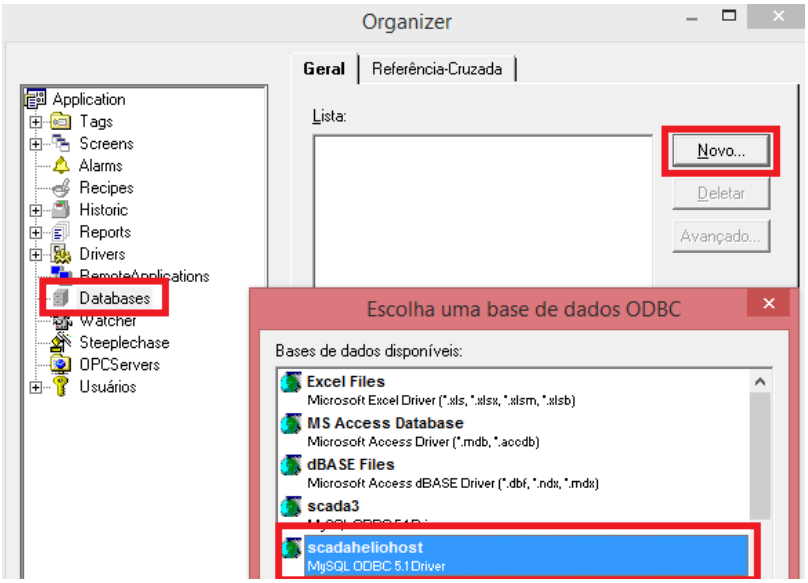


Figura 40 – Acessando conector ODBC pelo Elipse SCADA.

Os valores de potência ativa, potência reativa, data e hora serão as grandezas enviadas para o banco de dados. Assim, um *script* que roda na tela a cada um segundo, presente na Figura 41, enviará esses dados para a tabela "scada" criada no banco de dados. Portanto, os dados da tabela serão atualizados a cada um segundo.

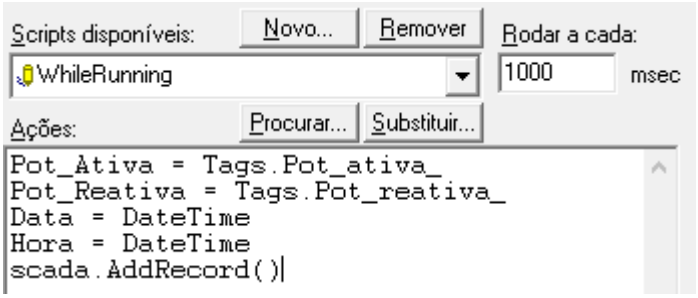


Figura 41 – *Script* de envio de variáveis ao banco de dados.

A administração do banco de dados pela *Web* é feita através do gerenciador PhpMyAdmin. A partir desse gerenciador é possível criar e remover base de dados, criar, remover e alterar tabelas e inserir, remover e editar campos. Na Figura 42 é possível visualizar a tela do PhpMyAdmin com o banco de dados criado para o projeto.

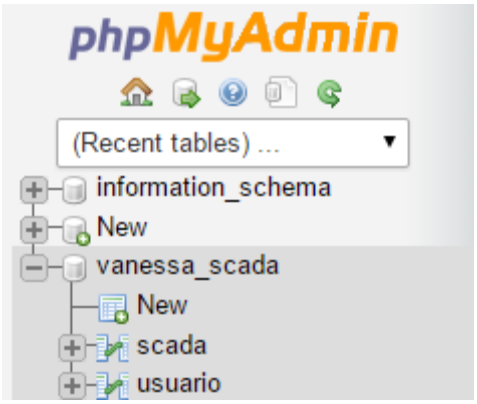


Figura 42 – Gerenciador PhpMyAdmin.

3.9 Desenvolvimento de uma Interface *Web* de Monitoramento

Para permitir um monitoramento remoto de algumas grandezas do sistema foi desenvolvida uma interface de monitoramento via *Web*. Tendo como intenção a criação de um site que apresente alguns valores do sistema em tempo real e possibilite mais algumas funcionalidades como visualização de histórico de dados e exportação de dados.

A página *Web* foi desenvolvida utilizando as linguagens de programação PHP (*Hypertext Preprocessor*) e HTML (*Hyper Text Markup Language*). Para desenvolver a programação e testar a mesma foi utilizado o *software* EasyPHP, já para enviar os arquivos para o servidor foi utilizado o *software* FileZilla. Para hospedagem do site na internet foi utilizado o serviço de hospedagem gratuita HelioHost.org, que possui suporte completo para linguagem PHP e para banco de dados MySQL. A página pode ser acessada no seguinte endereço www.vanessascada.heliohost.org.

Para acessar a interface *Web* de monitoramento, primeiramente, é necessário realizar o *login*. Foi criado no banco de dados uma tabela com usuários e senhas que possuem permissão para acessar a página. Em HTML foi programado um formulário para receber informações de usuário e senha, através do método *post*. Ao clicar no botão “entrar”, as informações digitadas passarão por uma autenticação, que foi programada em PHP para comparar os valores de usuário e senha recebidos com os valores presentes no banco de dados. Caso usuário e senha estejam corretos, o mesmo será direcionado para a página de supervisão e terá total acesso ao site. Caso contrário, o usuário receberá uma mensagem de erro e continuará na pagina de *login*, presente na Figura 43.

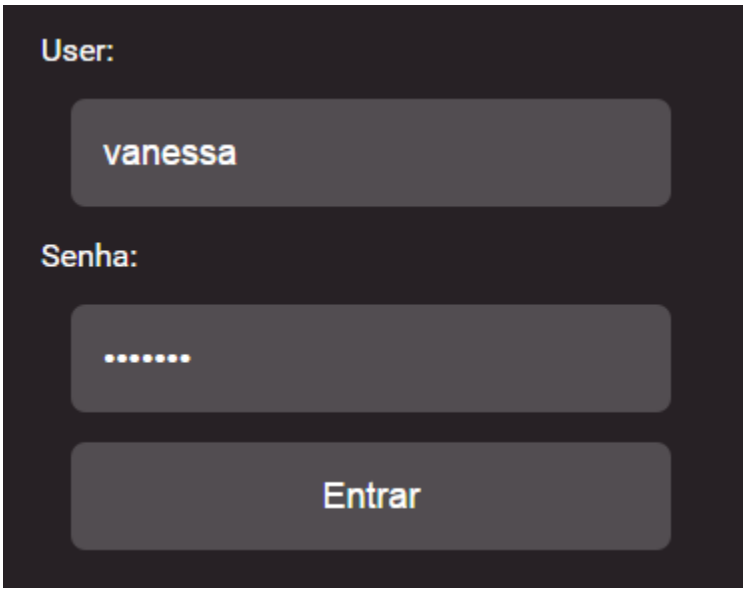


Figura 43 – Login da interface Web.

O site tem com página principal a página de supervisão, Figura 44. A página de supervisão foi programada em PHP para selecionar 15 valores de cada coluna da tabela do banco de dados, que possui conexão com o Elipse SCADA, e mostra-los em forma de tabela. Também foi programado em HTML que a página atualizará a cada 3 segundos. Assim, é possível monitorar em tempo real as variáveis do sistema.

Figura 44 – Site de monitoramento via *Web*.

Ao clicar na palavra “Histórico”, o usuário será direcionado a uma página onde poderá selecionar uma data para visualizar todos dados pertinentes a essa data. Para receber a data selecionada, foi programado em HTML um formulário que recebe a data escolhida pelo método *post*. Através da programação PHP a data escolhida é verificada no banco de dados e, caso exista dados nesse dia, será exibida na tela uma tabela com todos os dados da data inserida pelo usuário. A página para pesquisa do histórico pode ser visualizada na Figura 45.

Figura 45 – Página de histórico de dados.

Caso o usuário deseje exportar os dados referentes a alguma data, também existe a possibilidade de gerar uma tabela em formato *excel* para *download* através do site, basta clicar em “Gerar Tabela”, como mostrado na Figura 46.

Figura 46 – Página para gerar tabela de dados.

Para receber os dados (data e nome do arquivo a ser salvo), foi programado em HTML um formulário onde os dados inseridos são recebidos através do método *post*. Ao receber os valores, uma programação em PHP recebe a data e busca os dados dessa data no banco de dados. Após, é gerada uma planilha *excel* para *download* com o nome de arquivo escolhido pelo usuário e com os dados do banco de dados do dia selecionado.

Caso o usuário deseje sair da sessão é necessário clicar na palavra “Logout”, que direcionará o usuário a página de *login* novamente.

CAPÍTULO 4 RESULTADOS

4.1 Comunicação *Modbus* TCP/IP

Implementada as aplicações presentes nos Apêndices A e B no DSP e desenvolvido o sistema de supervisão, foi possível testar a comunicação *Modbus* TCP/IP. Portanto, para comprovar o tráfego de dados através do protocolo *Modbus* TCP/IP foi utilizado o *software* WireShark. Este programa monitora os pacotes transmitidos pelo dispositivo de comunicação do computador, permitindo a visualização da troca de dados entre o sistema supervisório e o DSP, como mostra a Figura 47.

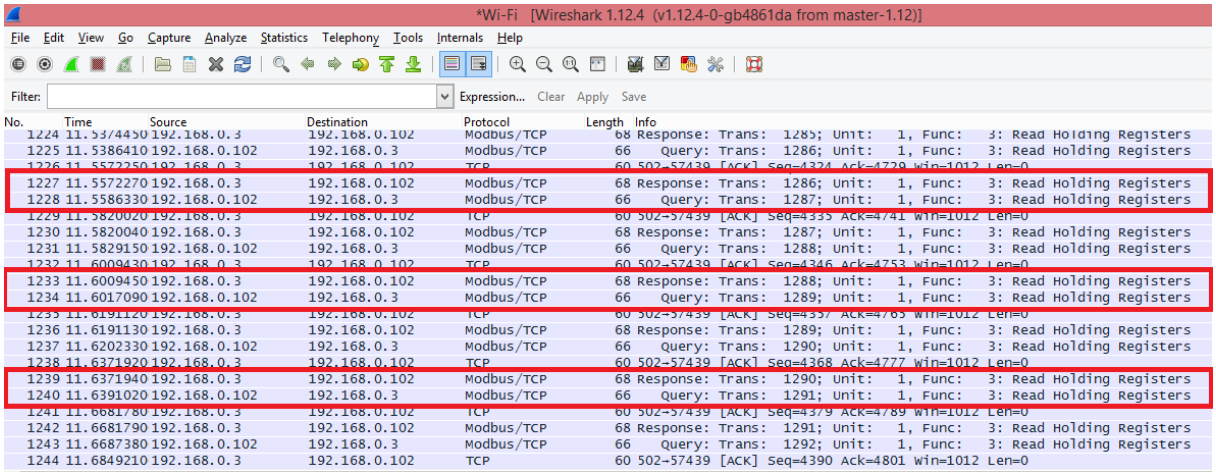


Figura 47 – Monitoração da troca de dados.

4.2 Resultados de Simulação

Nesta seção são apresentados os resultados de simulação do sistema. Para obtenção destes resultados foi utilizado o ambiente de simulação *Simulink/Matlab*, conforme Figura 48.

São apresentados resultados relativos a diferentes cargas (resistivas e capacitivas), bem como as correntes das cargas e os resultados dos cálculos de potência ativa instantânea e potência reativa instantânea.

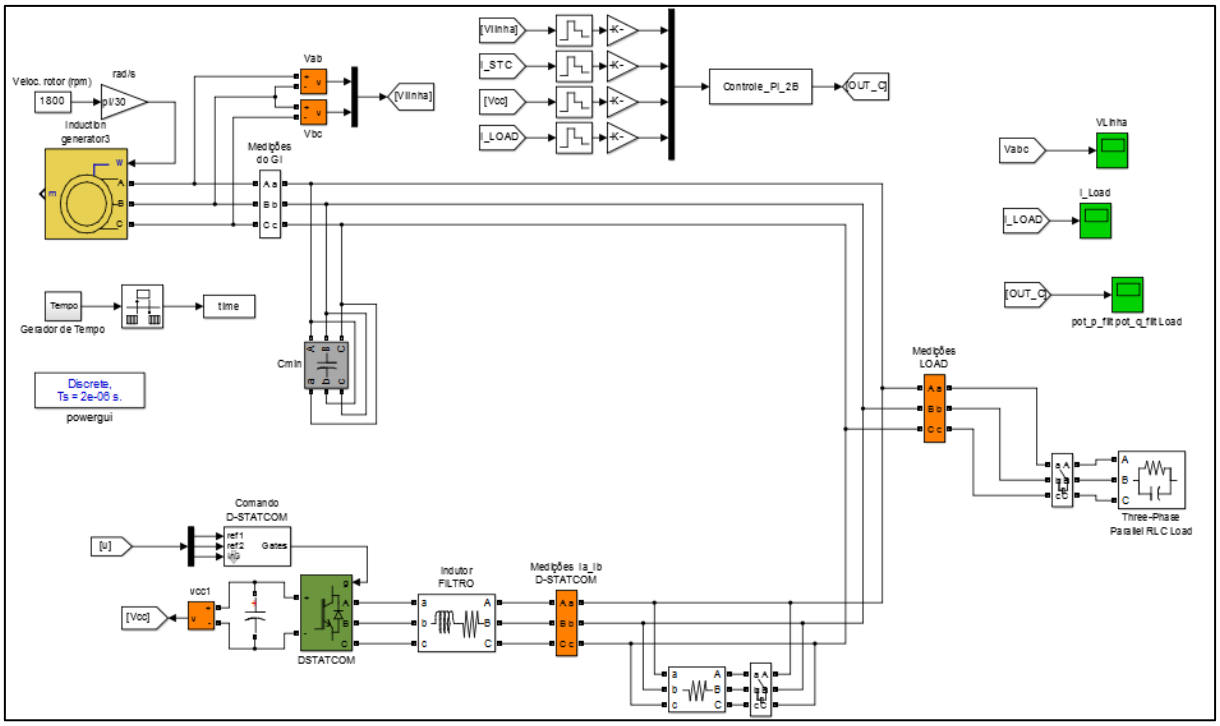


Figura 48 – Ambiente de simulação do sistema.

4.2.1 Carga Resistiva

Primeiramente, foram realizadas simulações apenas com as cargas resistivas ligadas em delta, a carga possui 300 W de potência. Foi atribuído ao gerador de indução uma tensão de 220 V_{rms}. Na Figura 49 é apresentada as formas de onda das correntes da carga I_a, I_b e I_c. Foram calculados os valores de potência ativa instantânea (linha azul do gráfico) e potência reativa instantânea (linha rosa do gráfico), como mostrado da Figura 50.

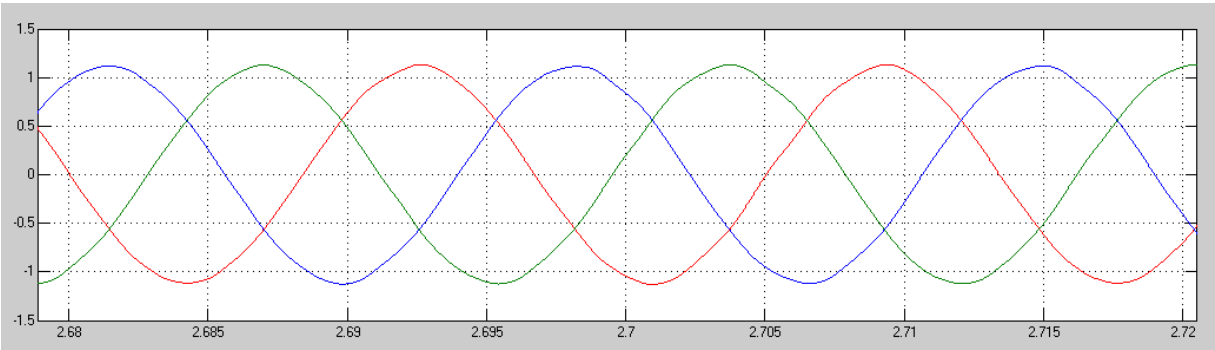


Figura 49 – Correntes da carga resistiva geradas a partir da simulação.

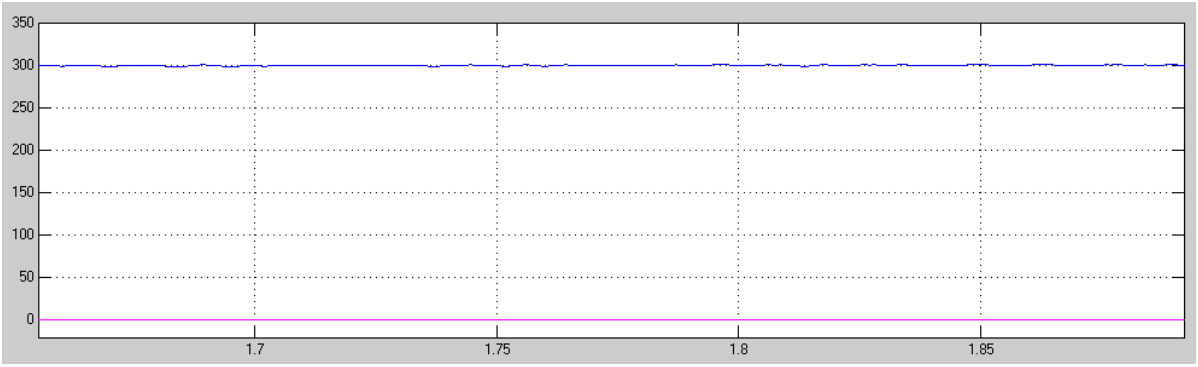


Figura 50 – Potência ativa e reativa geradas a partir da simulação.

4.2.2 Carga Resistiva e Capacitiva

Foram realizadas simulações com as cargas resistivas em paralelo com as cargas capacitivas, ambas ligadas em delta. A carga resistiva possui 300 W de potência e a carga capacitiva 12 µF. Foi atribuído ao gerador uma tensão de 220 V_{rms}. Na Figura 51 é apresentada as formas de onda das correntes da carga I_a, I_b e I_c. Foram calculados os valores de potência ativa instantânea (linha azul do gráfico) e potência reativa instantânea (linha rosa do gráfico), como mostrado da Figura 52.

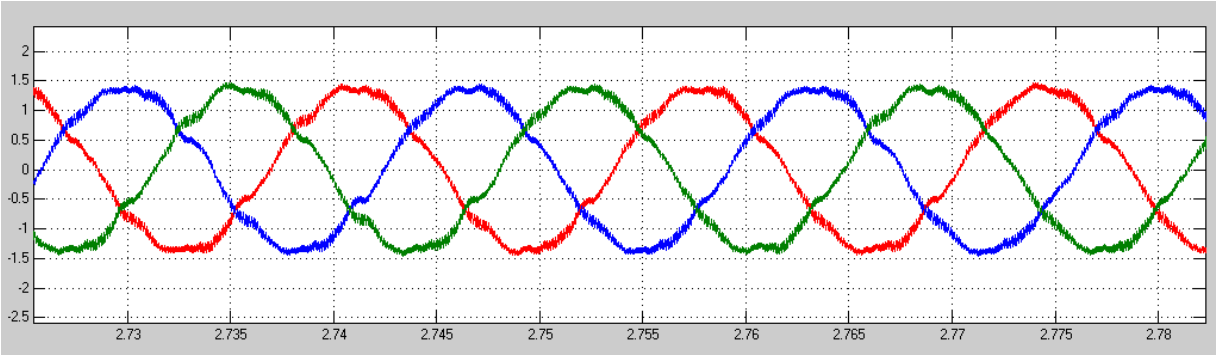


Figura 51 – Correntes da carga resistiva e capacitiva geradas a partir da simulação.

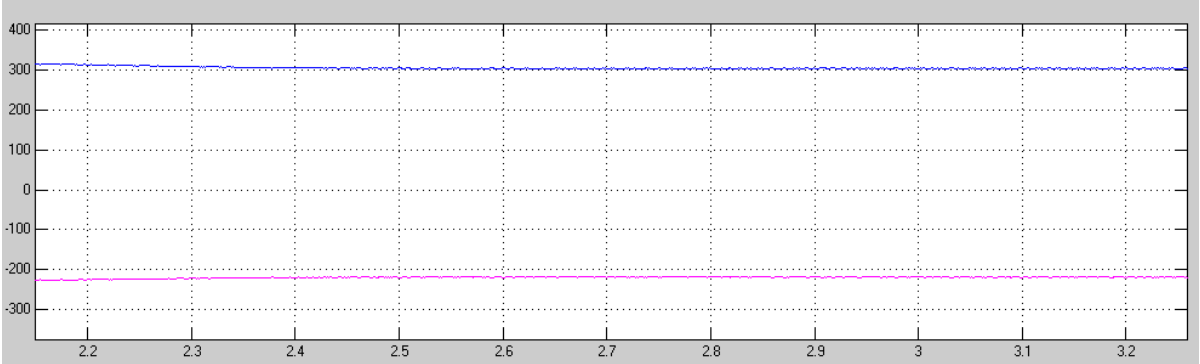


Figura 52 – Potência ativa e reativa geradas a partir de cargas resistivas e capacitivas.

4.3 Resultados Experimentais

Nesta seção são apresentados os resultados experimentais a partir de testes realizados em laboratório (Figura 53) e que utilizaram os seguintes materiais:

- Um variador de tensão trifásico para emular o funcionamento do gerador;
- As placas de tensão e de corrente para medição de dois valores de tensão e três de corrente;
- Uma carga resistiva composta de três lâmpadas de 220 V e 100W de potência cada;
- Uma carga capacitiva composta de três capacitores de 4 µF cada;
- Uma fonte de 15 V para alimentar a placa de interface;
- A placa de interface juntamente com a placa de adaptação conectada ao DSP *Concerto*;
- *Software Code Composer* para analisar a forma de onda do sinal que está sendo medido;
- Um roteador conectado via cabo Ethernet com o DSP para envio dos dados através do protocolo *Modbus* TCP/IP;
- Um computador com o sistema de supervisão que monitora em tempo real todas as variáveis medidas.

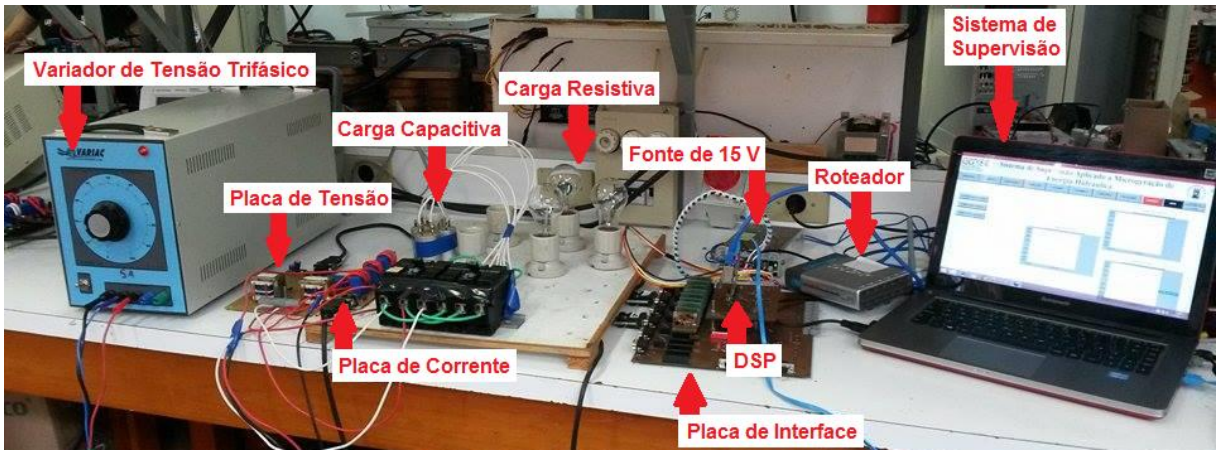


Figura 53 – Ambiente de testes experimentais.

São apresentados resultados relativos a diferentes cargas (resistivas e capacitivas), bem como resultados contemplando a aquisição de dados pelo DSP e através do sistema de supervisão. Nos resultados experimentais procura-se comprovar os resultados de simulação apresentados anteriormente.

4.3.1 Carga Resistiva

Primeiramente, foram realizados testes experimentais apenas com as cargas resistivas ligadas em delta. Foi atribuído ao variador de tensão trifásico uma tensão de 220 V_{rms}. As Figura 54 e Figura 55 apresentam as formas de onda das tensões de linha V_{ab} e V_{bc}, respectivamente, adquiridas pelo DSP através da placa de medição de tensão. Nota-se que as duas possuem valores de tensão pico a pico muito próximos que o valor pico a pico de 220 V_{rms}. Através da placa de medição de corrente também foi medido as correntes de carga I_a, I_b e I_c. Devido a limitações na resolução do gráfico do *Code Composer* para valores decimais muito pequenos, a corrente foi multiplicada por um ganho (neste caso 100 vezes) para ser visualizada com clareza em um gráfico, a Figura 56 mostra a corrente de carga I_a.

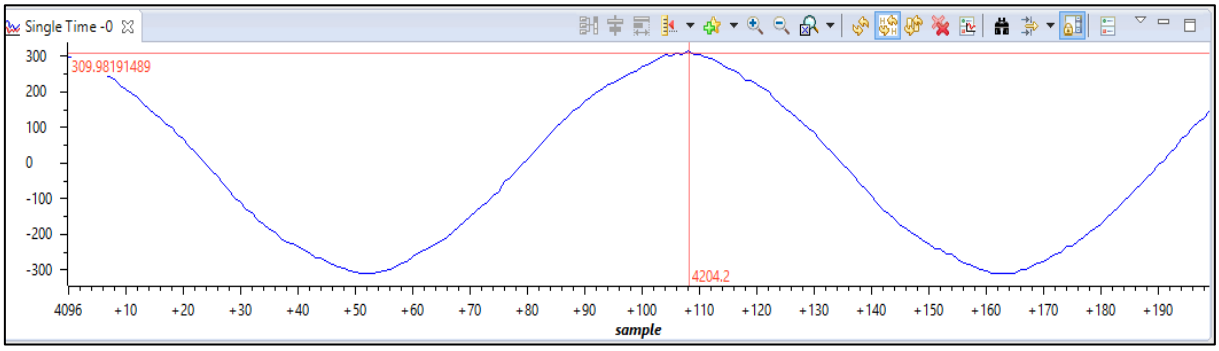


Figura 54 – Tensão de linha V_{ab} adquirida pelo DSP.

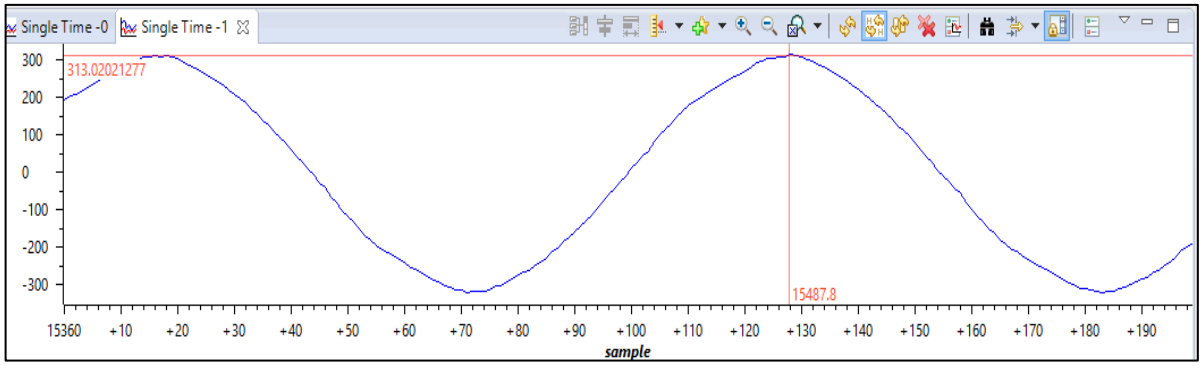


Figura 55 – Tensão de linha V_{bc} adquirida pelo DSP.

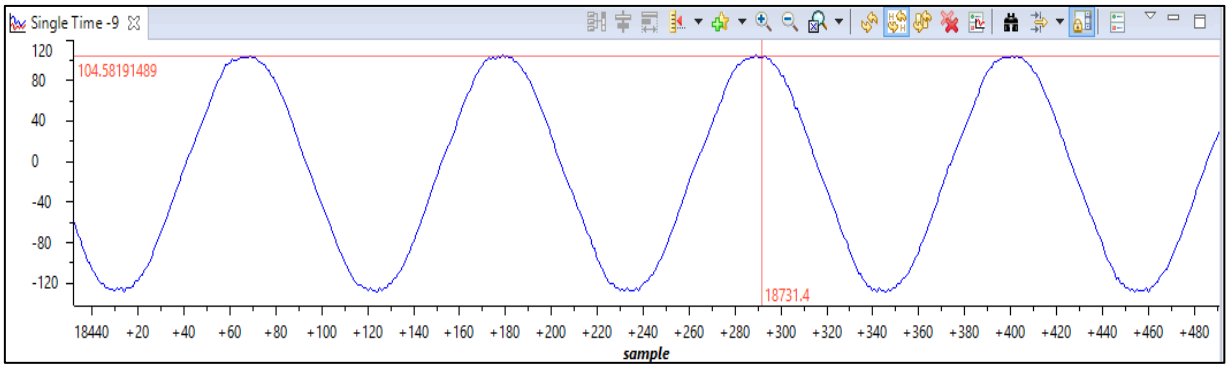


Figura 56 – Corrente de carga I_a adquirida pelo DSP (multiplicada por cem).

A partir dos valores de tensão e corrente foram calculados os valores de potência ativa instantânea (Figura 57), potência reativa instantânea (Figura 58) e potência aparente (Figura 59).

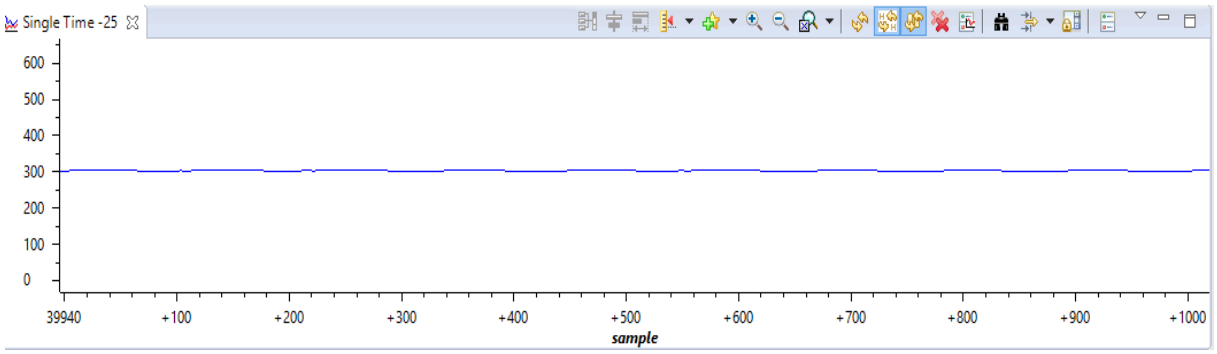


Figura 57 – Potência ativa calculada no DSP.

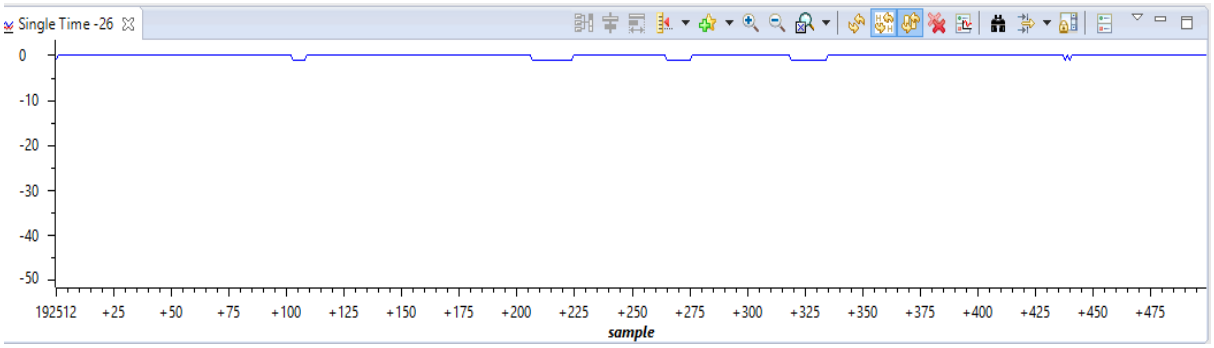


Figura 58 – Potência reativa calculada no DSP.

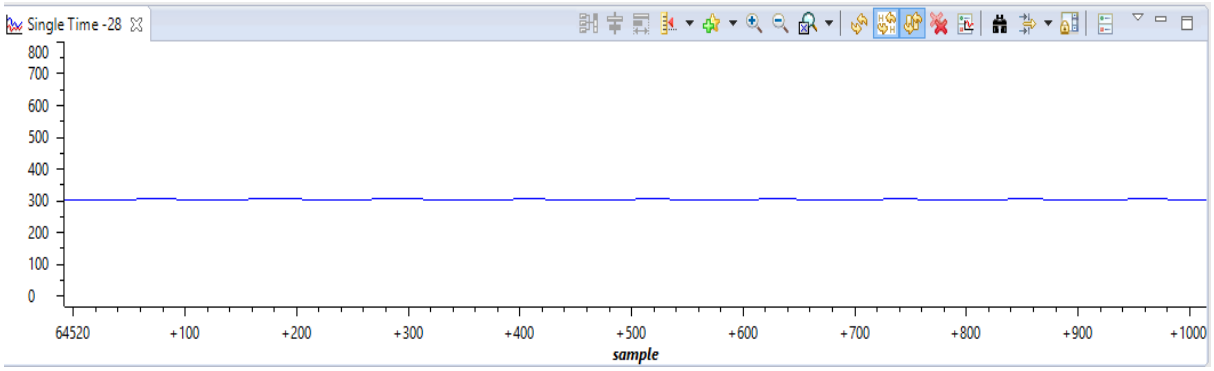


Figura 59 – Potência aparente calculada no DSP.

É possível visualizar através do sistema de supervisão os valores de tensão, como mostrado na Figura 60. O sistema de supervisão também permite a monitoração através de gráficos dos valores de tensão, conforme Figura 61.

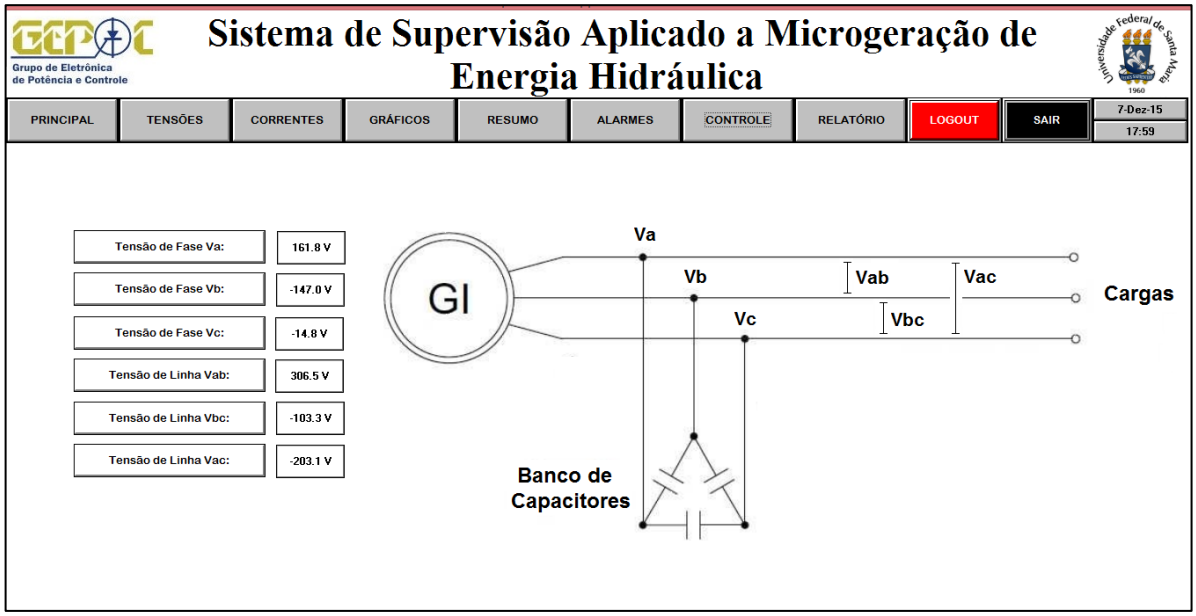


Figura 60 – Valores de tensão no sistema de supervisão.

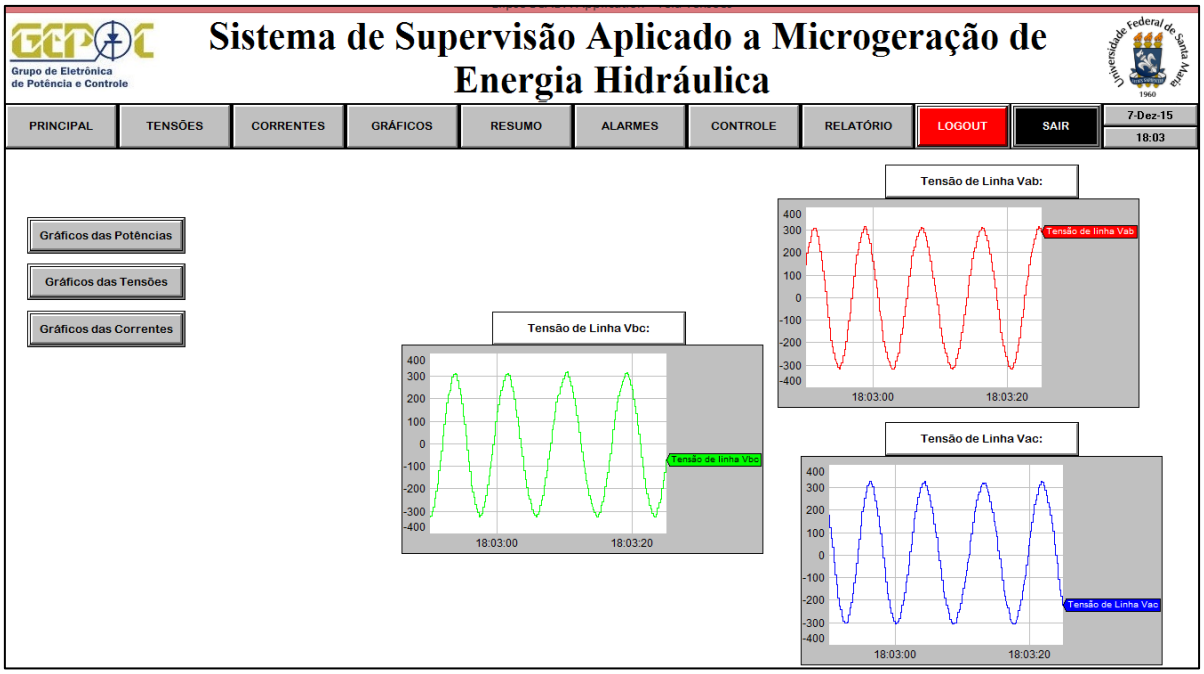


Figura 61 – Gráficos de tensões no sistema de supervisão.

O sistema de supervisão permite monitorar os valores de corrente da carga, do gerador e do DSTATCOM. Como está sendo medido apenas a corrente da carga, é possível visualizar os valores da mesma da Figura 62 e através de gráfico da Figura 63.

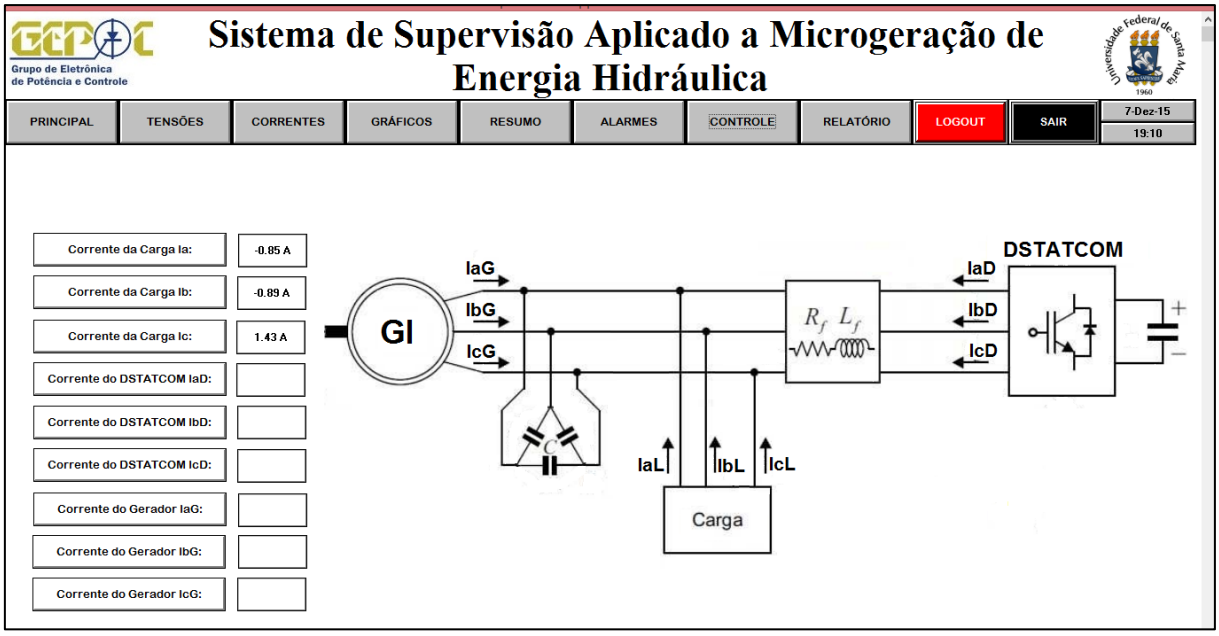


Figura 62 – Valores de corrente no sistema de supervisão

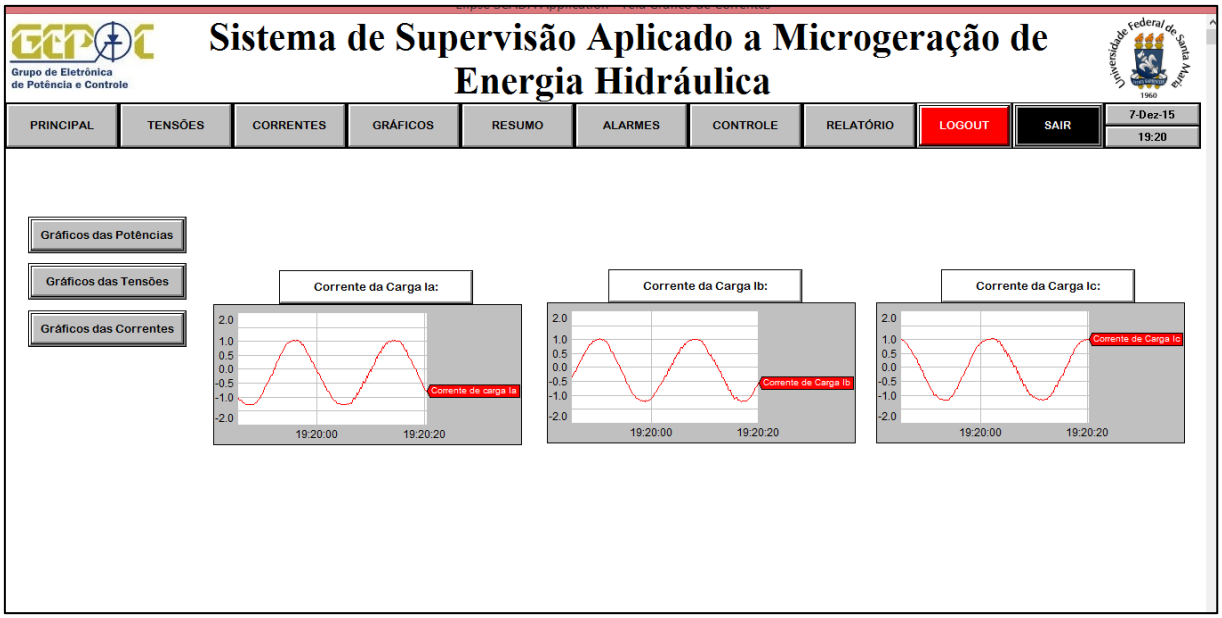


Figura 63 – Gráficos de corrente do sistema de supervisão.

Os valores de potência ativa, potência reativa, potência aparente e fator de potência podem ser monitorados em tempo real pelo sistema de supervisão, como mostrado na Figura 64 e Figura 65.

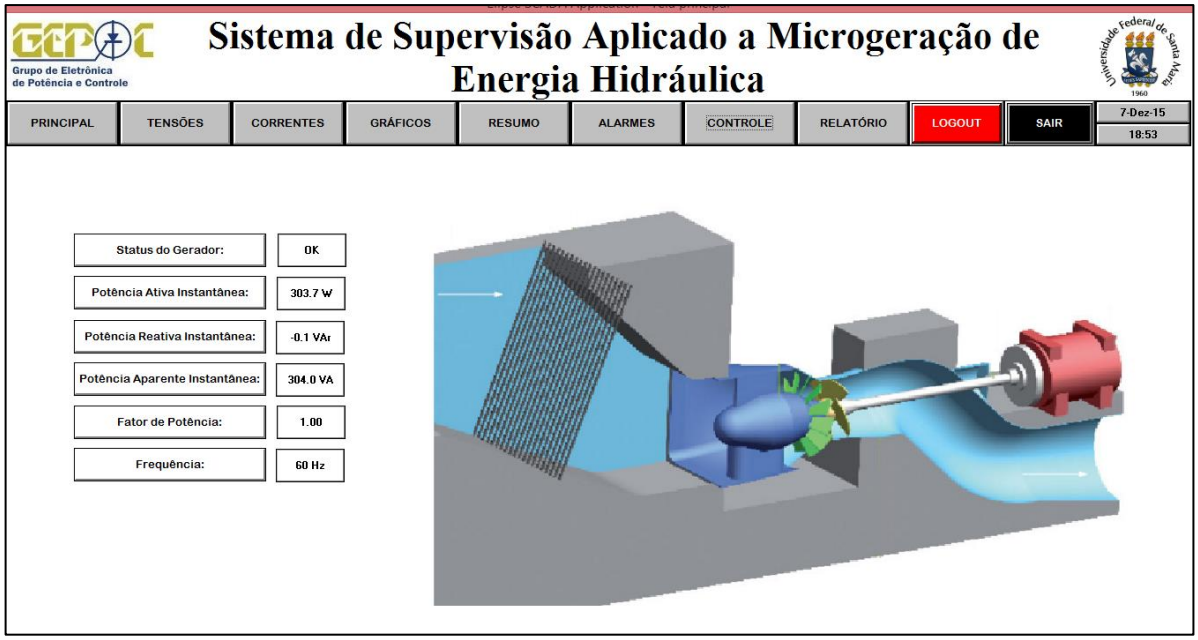


Figura 64 – Tela principal com valores experimentais.

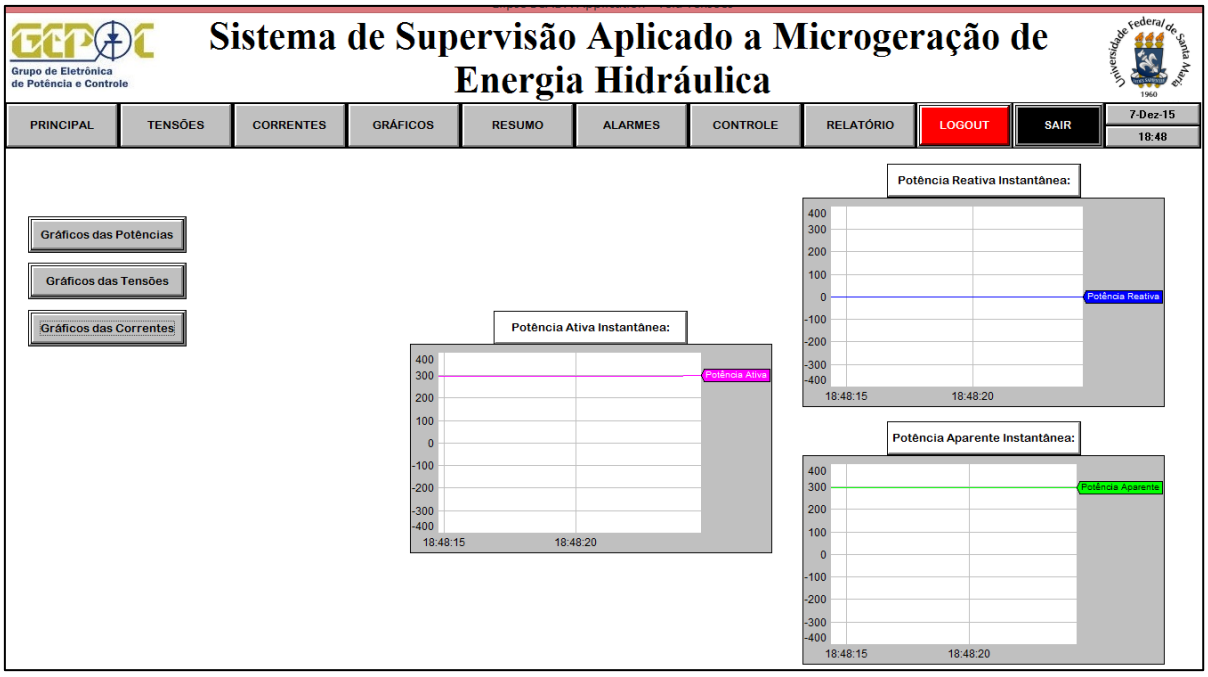
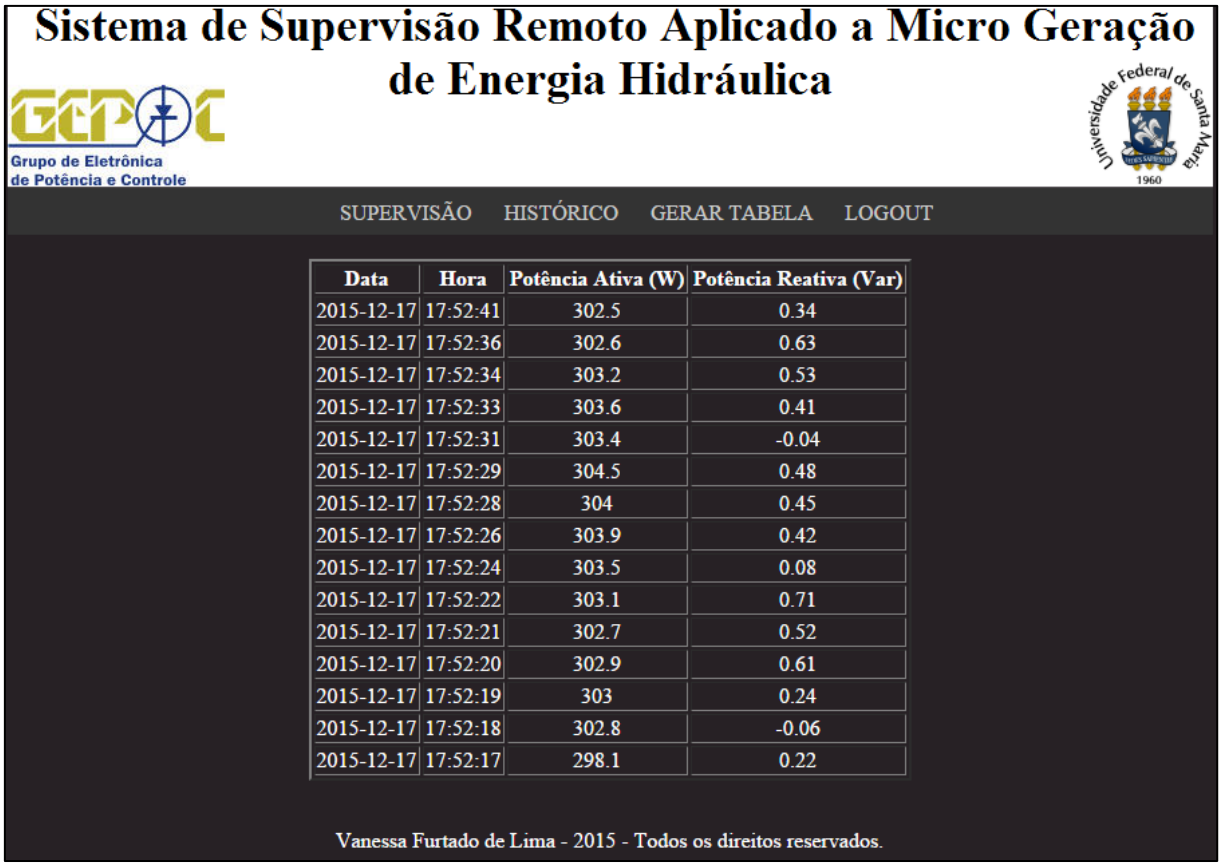


Figura 65 – Gráficos de potência do sistema de supervisão.

Através da interface *Web* também foi monitorado os valores de potência ativa e potência reativa em tempo real, conforme Figura 66.



Data	Hora	Potência Ativa (W)	Potência Reativa (Var)
2015-12-17	17:52:41	302.5	0.34
2015-12-17	17:52:36	302.6	0.63
2015-12-17	17:52:34	303.2	0.53
2015-12-17	17:52:33	303.6	0.41
2015-12-17	17:52:31	303.4	-0.04
2015-12-17	17:52:29	304.5	0.48
2015-12-17	17:52:28	304	0.45
2015-12-17	17:52:26	303.9	0.42
2015-12-17	17:52:24	303.5	0.08
2015-12-17	17:52:22	303.1	0.71
2015-12-17	17:52:21	302.7	0.52
2015-12-17	17:52:20	302.9	0.61
2015-12-17	17:52:19	303	0.24
2015-12-17	17:52:18	302.8	-0.06
2015-12-17	17:52:17	298.1	0.22

Figura 66 – Interface *Web* com monitoração em tempo real.

4.3.2 Carga Resistiva e Capacitiva

Foram realizados testes experimentais com as cargas resistivas em paralelo com as cargas capacitivas, ambas ligadas em delta. Foi atribuído ao variador de tensão trifásico uma tensão de 220 V_{rms}. A partir dos valores de tensão e corrente foram calculados os valores de potência ativa instantânea (Figura 67), potência reativa instantânea (Figura 68) e potência aparente (Figura 69).

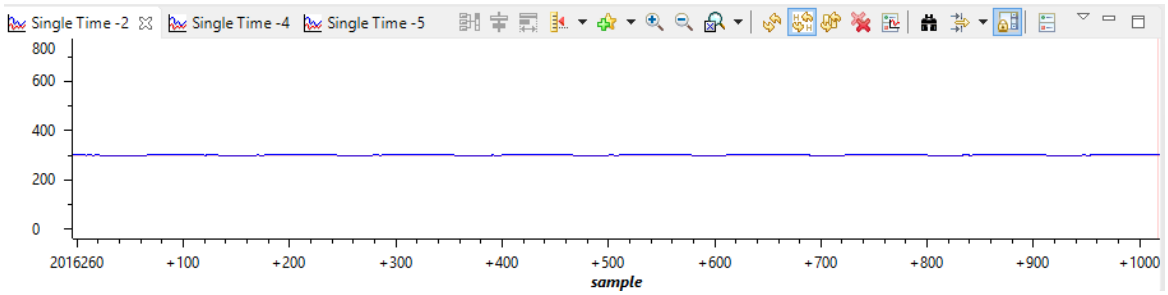


Figura 67 – Potência ativa com cargas resistivas e capacitivas.

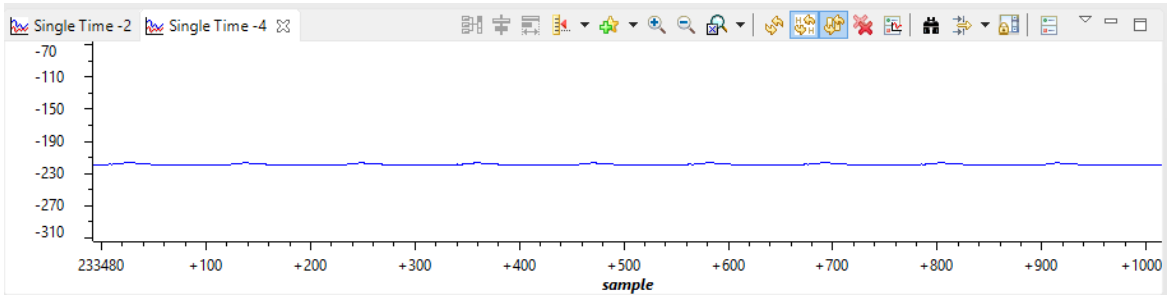


Figura 68 – Potência reativa com cargas resistivas e capacitivas.

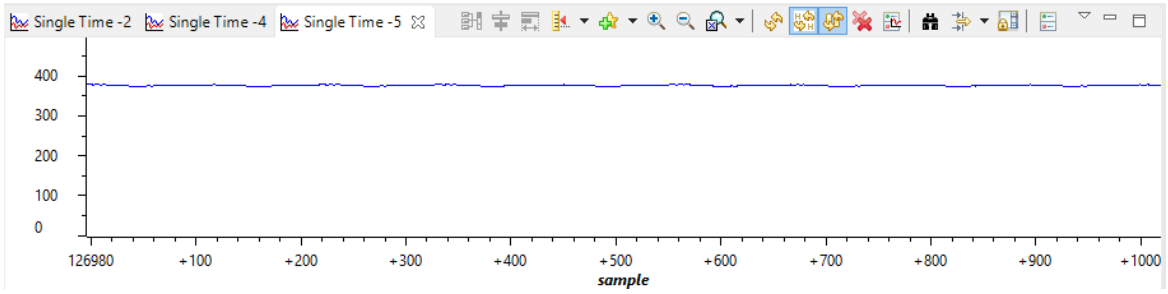


Figura 69 – Potência aparente com cargas resistivas e capacitivas.

O sistema de supervisão permite monitorar os valores de corrente da carga, do gerador e do DSTATCOM. Como está sendo medido apenas a corrente da carga, é possível visualizar os valores da mesma através de gráficos da Figura 70.

Os valores de potência ativa, potência reativa, potência aparente e fator de potência podem ser monitorados em tempo real pelo sistema de supervisão, como mostrado na Figura 71, onde é capturado o momento de transição onde as cargas capacitivas são acionadas.

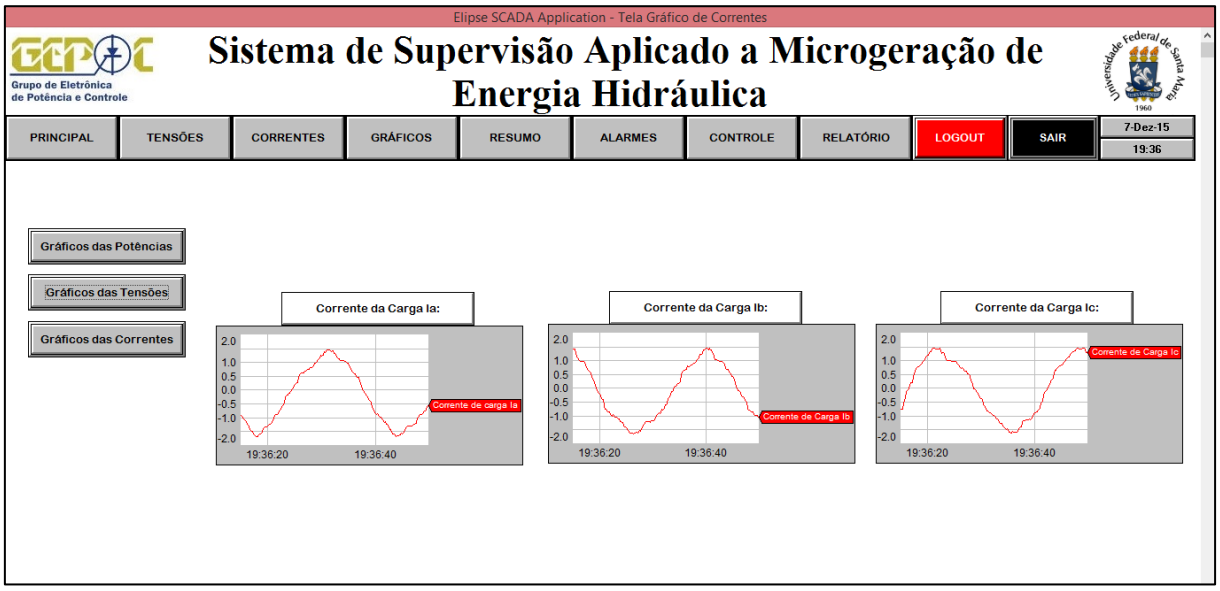


Figura 70 – Gráfico de corrente com carga resistiva e capacitiva do sistema de supervisão.

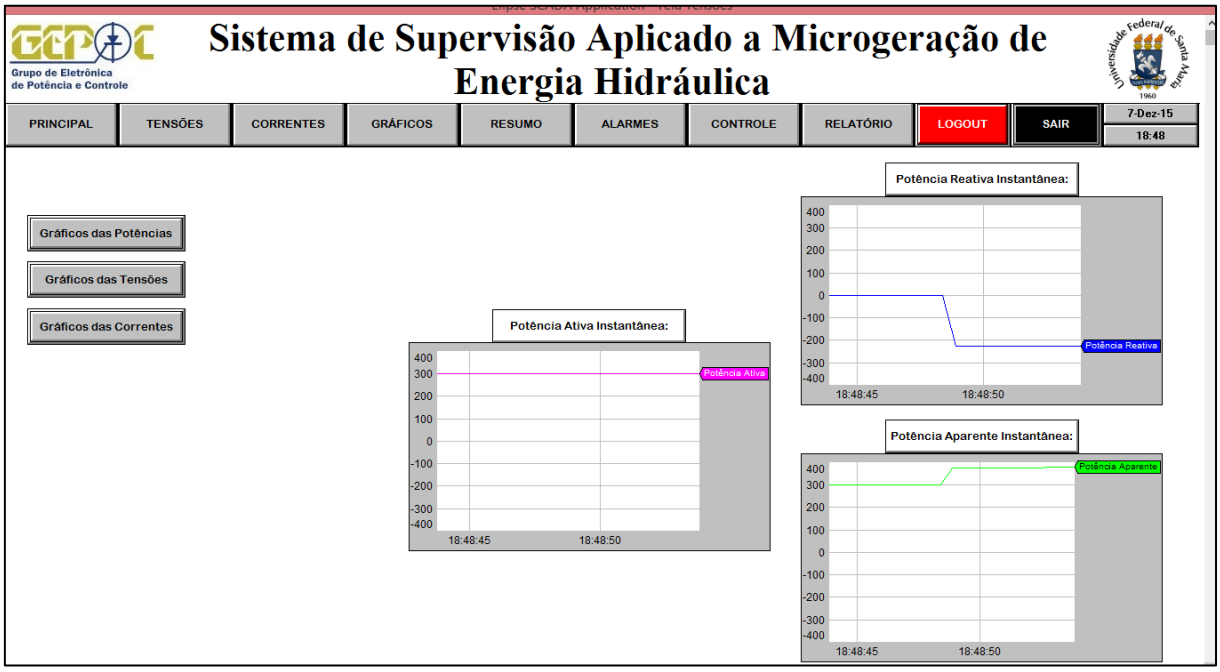
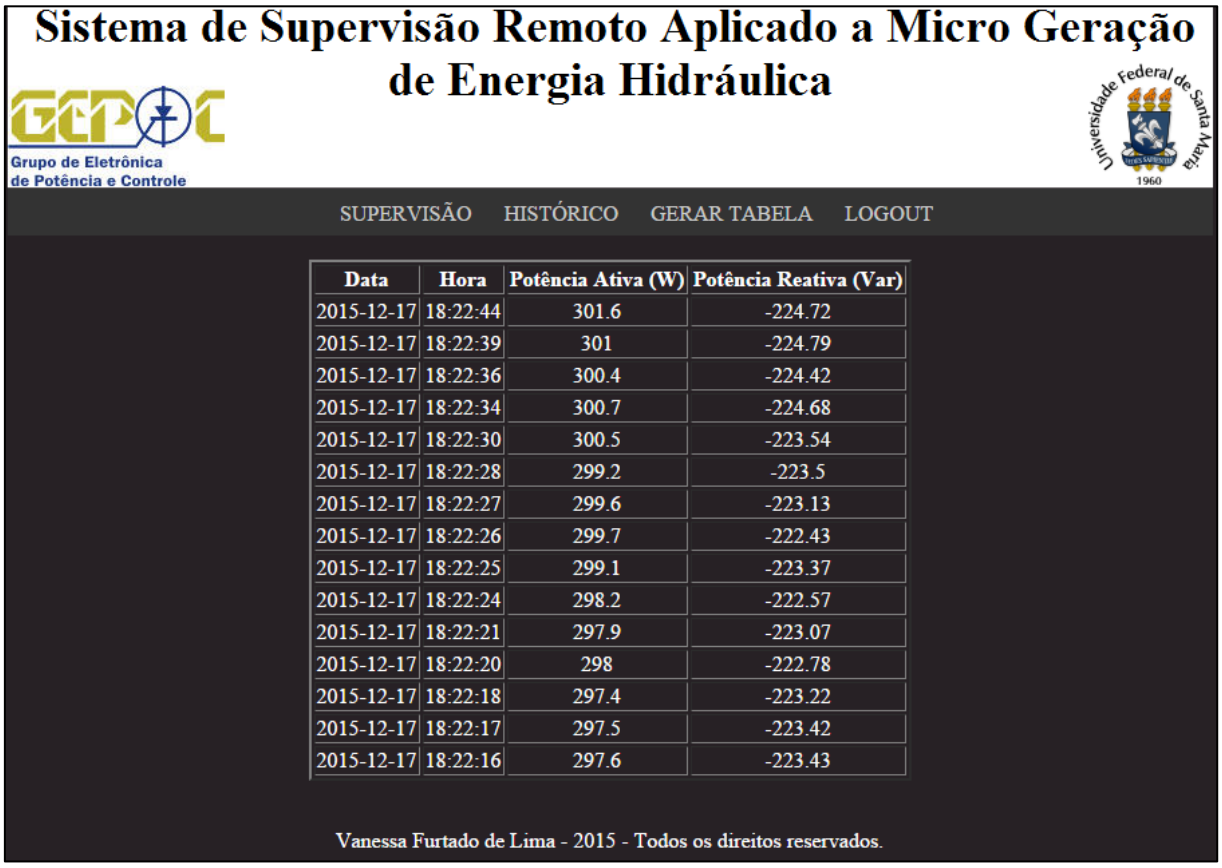


Figura 71 – Gráfico de potência com carga resistiva e capacitiva do sistema de supervisão.

Através da interface Web também foi possível monitorar os valores de potência ativa e potência reativa em tempo real, conforme figura.



Data	Hora	Potência Ativa (W)	Potência Reativa (Var)
2015-12-17	18:22:44	301.6	-224.72
2015-12-17	18:22:39	301	-224.79
2015-12-17	18:22:36	300.4	-224.42
2015-12-17	18:22:34	300.7	-224.68
2015-12-17	18:22:30	300.5	-223.54
2015-12-17	18:22:28	299.2	-223.5
2015-12-17	18:22:27	299.6	-223.13
2015-12-17	18:22:26	299.7	-222.43
2015-12-17	18:22:25	299.1	-223.37
2015-12-17	18:22:24	298.2	-222.57
2015-12-17	18:22:21	297.9	-223.07
2015-12-17	18:22:20	298	-222.78
2015-12-17	18:22:18	297.4	-223.22
2015-12-17	18:22:17	297.5	-223.42
2015-12-17	18:22:16	297.6	-223.43

Figura 72 – Inferface *Web* com monitoramento em tempo real com cargas resistivas e capacitivas.

CAPÍTULO 5 CRONOGRAMA DE ATIVIDADES

Na Tabela 2 encontra-se o cronograma de atividades realizadas.

Tabela 2 – Cronograma de atividades.

Atividade / Período	Agosto	Setembro	Outubro	Novembro	Dezembro
Revisão bibliográfica	X	X			
Estudo do DSP	X	X			
Estudo do Protocolo *Modbus*	X	X			
Implementação do protocolo *Modbus* TCP/IP no DSP Concerto F28M36x	X	X			
Estudo do funcionamento do sistema que emula uma microgeração de energia hidráulica	X	X			
Desenvolvimento e testes de uma placa de adaptação para conectar o DSP na placa de interface do sistema	X	X			
Implementação da comunicação *Modbus* TCP/IP entre o DSP e o sistema emulado		X	X		
Aquisição das grandezas de energia elétrica que estão sendo geradas utilizando o DSP		X	X		
Desenvolvimento do sistema de supervisão			X	X	
Integração do sistema supervisório com o DSP utilizando o protocolo *Modbus* TCP/IP.			X	X	
Desenvolvimento do sistema de monitoração via *Web*				X	X
Resultados de Simulação				X	X
Resultados Experimentais				X	X
Relatório Final				X	X

CAPÍTULO 6 CONCLUSÃO

O presente trabalho desenvolveu um sistema de supervisão para micro geração de energia hidráulica. Foram realizadas aquisições de dados e a implementação do protocolo de comunicação *Modbus* TCP/IP utilizando um DSP de alto desempenho para trocar dados com o sistema de supervisão desenvolvido. A partir disso, os dados coletados pelo sistema supervisório foram armazenados em um banco de dados em um servidor na *Web*. Através da linguagem de programação PHP e HTML foi criada uma interface de monitoramento via *Web*.

Com o auxilio de *software* foi possível monitorar a troca de dados e comprovar o funcionamento do protocolo utilizado para comunicação. Com a utilização do protocolo *Modbus* é possível desenvolver soluções em redes industriais capazes de atender a uma grande faixa de necessidades. Assim, avaliou-se que ao utilizar o protocolo de comunicação *Modbus* TCP/IP a troca de dados ocorreu de maneira robusta e confiável.

Através dos experimentos realizados com cargas resistivas e capacitivas foi analisada a veracidade e confiabilidade das medidas realizadas pelo DSP. Onde ao comparar os resultados simulados com os experimentais pode-se comprovar o correto funcionamento do sistema de aquisição e tratamento de medidas.

O sistema de supervisão também permitiu o monitoramento dos dados transmitidos em tempo real com uma interface amigável. O supervisório possui uma série de funcionalidades que tornam sua utilização mais completa e eficaz. Junto disso, a utilização do DSP para adquirir e transmitir as medidas do sistema emulado é uma importante ferramenta para teste de funcionamento tanto do sistema de supervisão quanto da rede de comunicação utilizada.

A interface *Web* permitiu com sucesso o monitoramento remoto do sistema, realizando em tempo real a atualização das variáveis medidas pelo sistema. A mesma também possibilitou um acesso ao histórico de dados e a funcionalidade de exportar os mesmos.

Para trabalhos futuros espera-se implementar o controle do sistema de micro geração de energia hidráulica no DSP, para assim, através do sistema de supervisão conseguir alterar em tempo real os parâmetros do controlador. Também é interessante migrar para uma plataforma gratuita de desenvolvimento do sistema de supervisão e desenvolver um aplicativo *Android* para monitoramento do mesmo.

REFERÊNCIAS

AGÊNCIA NACIONAL DE ENERGIA ELÉTRICA (ANEEL); **Micro e Mini Geração Distribuída: Sistema de Compensação de Energia Elétrica**, 2014, Caderno Temático. Disponível em <http://www.aneel.gov.br/biblioteca/downloads/livros/caderno-tematico-microeminigeracao.pdf>. Acesso em 24 Out. 2015.

AGÊNCIA NACIONAL DE ENERGIA ELÉTRICA (ANEEL); **Atlas de Energia Elétrica do Brasil**, 2008, 3ª Edição. Disponível em < http://www.aneel.gov.br/arquivos/PDF/atlas_par2_cap3.pdf>. Acesso em 19 Nov. 2015.

BOTTERÓN, F.; ANDRADE, M. A.; **Implementação em DSP de um Filtro Média Móvel**, Relatório de Processamento Digital de Sinais, 2002.

BOYER, S. A.; **SCADA – Supervisory Control and Data Acquisition,** 3ª Edição, The Instrumentation, Systems, and Automation Society, 2004.

COMUNIDADE HARDWARE; **Cabeamento, wireless, switches, roteadores e configuração**, 2009. Disponível em <http://www.hardware.com.br/comunidade/wireless-montar/958751/>. Acesso em 19 Nov. 2015.

DANEELS, A., W. SALTER; What is SCADA?, **International Conference on Acclelerator on Large Experimental Physic Control Systems**, Trieste, Itália, 2009.

DJIEV, S.; **Industrial Networks for Communication and Control,** Reading for Elements for Industrial Automation, Technical University, Sofia, Bulgaria, 2003, disponível em: http://anp.tu-sofia.bg/djiev/Networks.htm. Acesso em 26 abr. 2015.

DUMITRU, C, D; GLIGOR, A.; SCADA Based Software for Renewable Energy Management System, **Article in Procedia Economics and Finance**, 2012.

ELIPSE SCADA; **Manual do Usuário**, 2004. Disponível em <http://www.ebah.com.br/content/ABAAAARLsAF/elipse-scada-manual-br>. Acesso em 28 abr. 2015.

ELIPSE SOFTWARE; **Produtos – Sobre o Elipse E3**, 2015. Disponível em <http://www.elipse.com.br/port/e3.aspx>. Acesso em 20 Nov. 2015.

EMPRESA DE PESQUISA ENERGÉTICA (EPE); **Balanço Energético Nacional**, 2015. Disponível em <https://ben.epe.gov.br/downloads/Relatorio_Final_BEN_2015.pdf>. Acesso em 19 Nov. 2015.

FONSECA, M.; **Comunicação OPC – Uma abordagem prática**, 2002. Disponível em: < http://www.cpdee.ufmg.br/~seixas/PaginaSDA/Download/DownloadFiles/OPCMarcosFonseca.PDF>. Acesso em 06 Dez. 2015.

FREITAS, C.; **Protocolo Modbus: Fundamentos e Aplicações**, 2014. Disponível em: http://www.embarcados.com.br/protocolo-modbus-fundamentos-e-aplicacoes/. Acesso em 27 abr. 2015.

GUTIERREZ, R.M.V.; PAN, S.S.K.; **Complexo Eletrônico: Automação do Controle Industrial**. Disponível em: <http://www.bndes.gov.br/SiteBNDES/export/sites/default/bndes_pt/Galerias/Arquivos/conhecimento/bnset/set2807.pdf>. Acesso em 26 abr. 2015.

INFOWESTER, **Banco de Dados MySQL e PostgreSQL**, 2008. Disponível em: < http://www.infowester.com/postgremysql.php>. Acesso em 05 Dez. 2015.

MARRA, E. N.; POMÍLIO, J. A., Self-Excited Induction Generator Controlled by a VS-PWM Bidirectional Converter for Rural Applications, **IEEE Transactions on Industry Applications**, v. 35, n. 4, p. 877–883, 1999.

MOHD, A.; ORTJOHANN, E.; SCHMELTER, A.; HAMSIC, N.; MORTON, D.; Challenges in integrating distributed Energy storage systems into future smart grid, **IEEE Industrial Electronics**, 2008. ISIE 2008, p.1627–1632, 2008.

MORAES, C. C. de; CASTRUCCI, P. L.; **Engenharia de Automação Industrial**, 2ª.edição - LTC, 2007.

NATIONAL INSTRUMENTS (NI); **Artigos - Introdução ao Modbus,** 2014. Disponível em < http://www.ni.com/white-paper/7675/pt/>. Acesso em 29 Out. 2015.

PACONTROL; **Modbus Communication**, 2014. Disponível em: <http://www.pacontrol.com/Modbus.html>. Acesso em 28 Abr. 2015.

RADIO LOCMAN (RL); **Microcontrollers Usage: Texas Instruments F28M36x**, 2012. Disponível em: <http://www.radiolocman.com/news/new.html?di=144006>. Acesso em 29 de Out. 2015.

RAI, H. C. et al. "Voltage regulation of self-excited induction generator using passive elements". **Electrical Machines and Drives, 1993. Sixth International Conference** on (Conf. Publ. No. 376), p. 240–245, 1993.

REYNDERS, D.; MACKAY, S.; WRIGHT, E.; **Pratical Industrial Data Communications. Best Practice Techniques**. Newnes/Elsevier. 2005.

ROSÁRIO, J. M.; **Princípios de Mecatrônica,** 1ª Edição, São Paulo: Pearson, 2005.

SANTOS, H. G.; **Desenvolvimento de um supervisório modular para uma célula flexível de manufatura,** Dissertação de mestrado, UFSC, 2007.

SCHAF, F de M.; **Apostila de Redes Industriais**, ministrada na disciplina DPEE1052 do curso de Engenharia de Controle e Automação da UFSM, 2012/1.

SCHERER, L.G.; DE CAMARGO, R.F., "Frequency and voltage control of micro hydro power stations based on hydraulic turbine's linear model applied on induction generators", **Power Electronics Conference (COBEP)**, 2011 Brazilian , vol., no., pp.546,552, 11-15 Sept. 2011.

SILVA, R. E. F.; **Implementação de um Módulo de Supervisão para um Sistema de Detecção de Vazamentos em dutos de Petróleo**, Dissertação de mestrado, Universidade Federal do Rio Grande do Norte, Natal, 2009.

SILVA, A. P. G.; SALVADOR, M.; **"O que são sistemas supervisórios?"**, 2011. Disponível em <http://www.wectrus.com.br/artigos/sist_superv.pdf>. Acesso em 25 Out. 2015.

SINGH, B. et al; "Static synchronous compensator-variable frequency drive for voltage and frequency control of small-hydro driven self-excited induction generators system". **Generation, Transmission & Distribution, IET** , vol.8, no.9, pp.1528-1538, Sept. 2014.

SOLUÇÕES EM AUTOMAÇÃO INDUSTRIAL (SA); **InTouch HMI**, 2014. Disponível em <http://www.sa.online.pt/produtos/produtos_show.htm?idf=17&idp=94>. Acesso em 28 Out. 2015.

SOUZA, R. B.; **Uma arquitetra para sistemas supervisórios industriais e sua aplicação em processos de elevação artificial de petróleo**, Dissertação de mestrado, UFRN, 2005.

SOUZA, R. B.; A. A. D. MEDEIROS; J. M. A. NASCIMENTO; A. L. MAITELLI; H. P. GOMES; Um sistema supervisório para elevação artificial de petróleo, IBP, **Rio Oil & Gas Expo and Conference**, Rio de Janeiro, 2006.

TEXAS INSTRUMENTS, **F28M36x Concerto™ Microcontrollers datasheet**, 2012. Disponível em: <http://www.ti.com/lit/ds/symlink/f28m36h53b2.pdf>. Acesso em 27 Abr. 2015.

TEXAS INSTRUMENTS, **TMS320x2833x/2823x Enhanced Pulse Width Modulator (ePWM) Module**, 2009. Disponível em :<http://www.ti.com/lit/ug/sprug04a/sprug04a.pdf> Acesso em 28 Nov. 2015.

VIANNA, W. S.; **Sistema Scada Supervisório**. Campos dos Goytacazes. Rio de Janeiro, IFF, 2008.

YOUSSEF, K. H. et al. "A new method for voltage and frequency control of stand-alone self-excited induction generator using PWM converter with variable DC link voltage". 2008 **American Control Conference,** p. 2486–2491, 2008.

APÊNDICE A

Desenvolvimento da aplicação para o núcleo C28x, que é responsável pela aquisição de sinais a partir da entrada analógica do DSP, cálculo dos valores de tensão, corrente e potência, aplicação do filtro média móvel e envio de dados para o núcleo Cortex-M3.

```
//////////////////////////////////////////////////////////////////////////////
////////
// UNIVERSIDADE FEDERAL DE SANTA MARIA-UFSM
// GRUPO DE ELETRONICA DE POTENCIA E CONTROLE-GEPOC
//////////////////////////////////////////////////////////////////////////////
///////
// Código referente núcleo C28x do DSP Concerto F28M36x
// AUTORA: Vanessa Furtado de Lima
//////////////////////////////////////////////////////////////////////////////
//////
// DESCRIÇÃO:
// Este código realiza:
// > Aquisição de sinais a partir da entrada analógica do DSP
// > Cálculo dos valores de tensão, corrente e potência
// > Envio dos dados para o núcleo Cortex-M3
//////////////////////////////////////////////////////////////////////////////
/////
//CODE

/* Arquivos de cabeçalho XDCtools */
#include <xdc/std.h>
#include <xdc/runtime/Log.h>
#include <xdc/runtime/IHeap.h>
#include <xdc/runtime/Memory.h>
#include <xdc/runtime/Error.h>
#include <xdc/runtime/System.h>
#include <xdc/cfg/global.h>

/* Arquivos de cabeçalho IPC */
#include <ti/ipc/MessageQ.h>
#include <ti/ipc/MultiProc.h>

/* Arquivos de cabeçalho BIOS  */
#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/heaps/HeapBuf.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Semaphore.h>

#include <string.h>
#include "demo.h"
#include <math.h>

#include "DSP28x_Project.h"// Device Headerfile and Examples Include File

#define BUF_SIZE   100  // Sample buffer size
#define  PERIOD         7500 // Período do PWM - 10Khz
#define  DUTY_CYCLE_A   3750
#define  DUTY_CYCLE_B   3750
```

```
// Configure which ePWM timer interrupts are enabled at the PIE level:
// 1 = enabled,  0 = disabled
#define PWM1_INT_ENABLE  1

Void adc_fxn(UArg arg);

// Global variables used in this example
UInt16  adcResult;
UInt16  buffer[BUF_SIZE];
Uint32  EPwm1TimerIntCount, interrupcao = 0, count_led = 0;
// Variáveis Supervisão MCH
float Tensao_va_fase, Tensao_vb_fase, Tensao_vc_fase, Tensao_valinha,
Tensao_vblinha, Tensao_vclinha, Ia_carga, Ib_carga, Ic_carga,
            Pot_ativa, Pot_reativa, Pot_aparente, Ia, Ib, Ic, Fat_potencia ;
long int Tensao_vb_fase_i, Tensao_vc_fase_i, Tensao_valinha_i, Tensao_vblinha_i,
Tensao_vclinha_i, Ia_carga_i, Ib_carga_i, Ic_carga_i,
Pot_ativa_i, Pot_reativa_i, Pot_aparente_i, Ia_i, Ib_i, Ic_i, Fat_potencia_i;
long int Tensao_va_fase_i;

float Ia_carga_off, Ib_carga_off, Ic_carga_off, Ia_off, Ib_off, Ic_off,
Tensao_valinha_off,
Tensao_vblinha_off;


int BUFFER_SIZE1=1024;
int buffer01[1024], buffer02[1024], buffer03[1024], buffer04[1024],
buffer05[1024], buffer06[1024], buffer07[1024];
long int Pot_ativa_med, Pot_ativa_med_aux, Pot_reativa_med, Pot_reativa_med_aux;
long int i;
int k;
int pwmteste;
// filtro media móvel
int filtro_ativo [200], filtro_reativo[200], filtro_aparente[200];
long int Pot_ativa_aux, Pot_reativa_aux, Pot_aparente_aux;
float Pot_ativa_filt, Pot_reativa_filt;
float Pot_aparente_filt;
int z, j;
int buffer08[1024];

// Rotina de comunicação com MCU Cortex-M3 do DSP Concerto por meio do módulo
MESSAGE_Q
/*
 *  ======== tsk0_func ========
 *  Allocates a message and ping-pongs the message around the processors.
 *  A local message queue is created and a remote message queue is opened.
 *  Messages are sent to the remote message queue and retrieved from the
 *  local MessageQ.
 */
Void tsk0_func(UArg arg0, UArg arg1)
{
    MessageQ_Msg     msg;
    MessageQ_Handle  messageQ;
    MessageQ_QueueId remoteQueueId;
    int              status;
    //unsigned short   msgId = 0;
    Ptr              buf;
    HeapBuf_Handle   heapHandle;
    HeapBuf_Params   hbparams;
    SizeT            blockSize;
```

```
unsigned int     numBlocks;
Error_Block      eb;

/* Compute the blockSize & numBlocks for the HeapBuf */
numBlocks = 16;
blockSize = sizeof(TempMsg);

/* Alloc a buffer from the default heap */
buf = Memory_alloc(0, numBlocks * blockSize, 0, NULL);

/*
 *  Create the heap that is used for allocating MessageQ messages.
 */
Error_init(&eb);
HeapBuf_Params_init(&hbparams);
hbparams.align          = 0;
hbparams.numBlocks      = numBlocks;
hbparams.blockSize      = blockSize;
hbparams.bufSize        = numBlocks * blockSize;
hbparams.buf            = buf;
heapHandle = HeapBuf_create(&hbparams, &eb);
if (heapHandle == NULL) {
    System_abort("HeapBuf_create failed\n" );
}

/* Register default system heap with MessageQ */
MessageQ_registerHeap((IHeap_Handle)(heapHandle), HEAPID);

/* Create the local message queue */
messageQ = MessageQ_create(C28QUEUENAME, NULL);
if (messageQ == NULL) {
    System_abort("MessageQ_create failed\n" );
}

/* Open the remote message queue. Spin until it is ready. */
do {
    status = MessageQ_open(M3QUEUENAME, &remoteQueueId);
    /*
     *  Sleep for 1 clock tick to avoid inundating remote processor
     *  with interrupts if open failed
     */
    if (status < 0) {
        Task_sleep(1);
    }
} while (status < 0);

/*
 *  Wait for a message from the M3 processor and
 *  send it back after converting to Fahrenheit.
 */
while (1) {
    /* Get a message */
status = MessageQ_get(messageQ, &msg, MessageQ_FOREVER);
if (status < 0) {
   System_abort("This should not happen since timeout is forever\n");
}
     /* Get the message id */
```

```
    // Variáveis MCH

            ((TempMsg *)msg)->Tensao_va_fase =  Tensao_va_fase_i;
            ((TempMsg *)msg)->Tensao_vb_fase = Tensao_vb_fase_i;
            ((TempMsg *)msg)->Tensao_vc_fase = Tensao_vc_fase_i;
            ((TempMsg *)msg)->Tensao_valinha = Tensao_valinha_i;
            ((TempMsg *)msg)->Tensao_vblinha = Tensao_vblinha_i;
            ((TempMsg *)msg)->Tensao_vclinha = Tensao_vclinha_i;
            ((TempMsg *)msg)->Ia_carga = Ia_carga_i;
            ((TempMsg *)msg)->Ib_carga = Ib_carga_i;
            ((TempMsg *)msg)->Ic_carga = Ic_carga_i;
            ((TempMsg *)msg)->Pot_ativa_filt = Pot_ativa_i;
            ((TempMsg *)msg)->Pot_reativa_filt = Pot_reativa_i;
            ((TempMsg *)msg)->Pot_aparente_filt = Pot_aparente_i;
            ((TempMsg *)msg)->Ia = Ia_i;
            ((TempMsg *)msg)->Ib = Ib_i;
            ((TempMsg *)msg)->Ic = Ic_i;
            ((TempMsg *)msg)->Fat_potencia = Fat_potencia_i;

        status = MessageQ_put(remoteQueueId, msg);
        if (status < 0) {
           System_abort("MessageQ_put had a failure/error\n");
        }
    }
}


// Função MAIN que executa inicialização do DSP, parametrização do ADC e PWM e
aquisição
// de dados
/*
 *  ======== main ========
 */

Void main()
{
    // Initialize System Control for Control and Analog Subsytems
    // Enable Peripheral Clocks
    // This example function is found in the F28M35x_SysCtrl.c file.
    InitSysCtrl();

    // Step 2. Initialize GPIO:
    // This example function is found in the F28M36x_Gpio.c file and
    // illustrates how to set the GPIO to it's default state.
    InitGpio();


    EALLOW;
    //LED's Control Card
    GpioG1CtrlRegs.GPADIR.bit.GPIO31  = 1;  //Set as output  //LEDs

    GpioG1CtrlRegs.GPAMUX2.bit.GPIO31 = 0;

    GpioG1CtrlRegs.GPADIR.bit.GPIO0  = 1;  //Set as output  PWM

    GpioG1CtrlRegs.GPAMUX1.bit.GPIO0 = 1 ;

    EDIS;
```

```
    GpioG1DataRegs.GPADAT.bit.GPIO31 = 1;
    InitFlash();

    // Initialize all the Device Peripherals:
    // This function is found in F28M35x_InitPeripherals.c
    InitAdc1();
    //Adc2-vanessa
    InitAdc2();
    //adc2-fim
    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;      // Stop all the TB clocks
    EDIS;

    EALLOW;

    SysCtrlRegs.PCLKCR1.bit.EPWM1ENCLK = 1;  // ePWM1
    SysCtrlRegs.PCLKCR1.bit.EPWM2ENCLK = 1;  // ePWM2
    SysCtrlRegs.PCLKCR1.bit.EPWM3ENCLK = 1;  // ePWM3
    SysCtrlRegs.PCLKCR1.bit.EPWM4ENCLK = 1;  // ePWM4
    SysCtrlRegs.PCLKCR1.bit.EPWM5ENCLK = 1;  // ePWM5
    SysCtrlRegs.PCLKCR1.bit.EPWM6ENCLK = 1;  // ePWM6

    EDIS;


    EALLOW;
    //Assumes ePWM1 clock is already enabled in InitSysCtrl();
    //Set event triggers (SOCA) for ADC SOC1
    EPwm1Regs.ETSEL.bit.SOCAEN  = 1;          // Enable SOC on A group
    EPwm1Regs.ETSEL.bit.SOCASEL = ET_CTR_PRDZERO;    // Select SOC from CMPA on
upcount
    EPwm1Regs.ETPS.bit.SOCAPRD  = ET_1ST;        // Generate pulse on every 1st
event

    //Time-base registers
    EPwm1Regs.TBPRD = PERIOD;                     // Set timer period, PWM
frequency = 1/period
    EPwm1Regs.TBPHS.half.TBPHS = 0;               // Time-Base Phase Register
    EPwm1Regs.TBCTL.bit.PRDLD = TB_SHADOW;      // Set Immediate load
    EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;   // Count-up mode: used for
asymmetric PWM
    EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;       // Disable phase loading
    EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO;
    EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;
    EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;

    //Setup shadow register load on ZERO
    EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;   // load on CTR=Zero
    EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;   // load on CTR=Zero

    //Set actions
    EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;     // set actions for EPWM1A
    //EPwm1Regs.AQCTLA.bit.CAD = AQ_CLEAR;
    EPwm1Regs.AQCTLA.bit.CAD = AQ_SET;

    //Configure Dead Time
    EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;   // enable Dead-band module
```

```
    EPwm1Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;    // Active Hi complementary
    EPwm1Regs.DBFED = 50;         // FED = 50 TBCLKs
    EPwm1Regs.DBRED = 50;        // RED = 50 TBCLKs

    EPwm1Regs.CMPA.half.CMPA = DUTY_CYCLE_A;      // Set duty 50% initially


    EDIS;

    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;         // Start all the timers synced
    EDIS;

    EALLOW;

    // Configure ADC

    Adc1Regs.ADCCTL2.bit.ADCNONOVERLAP = 1;     // Enable non-overlap mode i.e.
conversion and future sampling events dont overlap
    Adc1Regs.ADCCTL1.bit.INTPULSEPOS    = 1;     // ADCINT1 trips after AdcResults
latch
    Adc1Regs.INTSEL1N2.bit.INT1E        = 1;     // Enabled ADCINT1
    Adc1Regs.INTSEL1N2.bit.INT1CONT     = 0;     // Disable ADCINT1 Continuous mode
    Adc1Regs.INTSEL1N2.bit.INT1SEL      = 0;     // setup EOC0 to trigger ADCINT1
to fire

    //Adc2-vanessa
    Adc2Regs.ADCCTL2.bit.ADCNONOVERLAP = 1;     // Enable non-overlap mode i.e.
conversion and future sampling events dont overlap
    Adc2Regs.ADCCTL1.bit.INTPULSEPOS    = 1;     // ADCINT1 trips after AdcResults
latch
    Adc2Regs.INTSEL1N2.bit.INT1E        = 1;     // Enabled ADCINT1
    Adc2Regs.INTSEL1N2.bit.INT1CONT     = 0;     // Disable ADCINT1 Continuous mode
    Adc2Regs.INTSEL1N2.bit.INT1SEL      = 0;     // setup EOC0 to trigger ADCINT1
to fire
    //Adc2-fim

    Adc1Regs.ADCSAMPLEMODE.bit.SIMULEN0 = 1;    // Simultaneous sampling enable
for SOC0/SOC1
    Adc1Regs.ADCSAMPLEMODE.bit.SIMULEN2 = 1;    // Simultaneous sampling enable
for SOC2/SOC3
    Adc1Regs.ADCSAMPLEMODE.bit.SIMULEN4 = 1;    // Simultaneous sampling enable
for SOC4/SOC5
    Adc1Regs.ADCSAMPLEMODE.bit.SIMULEN6 = 1;    // Simultaneous sampling enable
for SOC6/SOC7
    Adc1Regs.ADCSAMPLEMODE.bit.SIMULEN8 = 1;    // Simultaneous sampling enable
for SOC8/SOC9
    Adc1Regs.ADCSAMPLEMODE.bit.SIMULEN10 = 1;   // Simultaneous sampling enable
for SOC10/SOC11
    //OUTROS ADC1- vanessa
    Adc1Regs.ADCSAMPLEMODE.bit.SIMULEN12 = 1;   // Simultaneous sampling enable
for SOC12/SOC13
    Adc1Regs.ADCSAMPLEMODE.bit.SIMULEN14 = 1;   // Simultaneous sampling enable
for SOC14/SOC15
    //OUTRO ADC 1- fim

    //Adc2-vanessa
    Adc2Regs.ADCSAMPLEMODE.bit.SIMULEN0 = 1;    // Simultaneous sampling enable
for SOC0/SOC1
```

```
    Adc2Regs.ADCSAMPLEMODE.bit.SIMULEN2 = 1;    // Simultaneous sampling enable
for SOC2/SOC3
    Adc2Regs.ADCSAMPLEMODE.bit.SIMULEN4 = 1;    // Simultaneous sampling enable
for SOC4/SOC5
    Adc2Regs.ADCSAMPLEMODE.bit.SIMULEN6 = 1;    // Simultaneous sampling enable
for SOC6/SOC7
    Adc2Regs.ADCSAMPLEMODE.bit.SIMULEN8 = 1;    // Simultaneous sampling enable
for SOC8/SOC9
    Adc2Regs.ADCSAMPLEMODE.bit.SIMULEN10 = 1;   // Simultaneous sampling enable
for SOC10/SOC11
    Adc2Regs.ADCSAMPLEMODE.bit.SIMULEN12 = 1;   // Simultaneous sampling enable
for SOC12/SOC13
    Adc2Regs.ADCSAMPLEMODE.bit.SIMULEN14 = 1;   // Simultaneous sampling enable
for SOC14/SOC15
    //Adc2-fim

    //Setting up the trigger source
    AnalogSysctrlRegs.TRIG1SEL.all  = 5;      // Assigning EPWM1SOCA to ADC
TRIGGER 1 of the ADC module

    Adc1Regs.ADCSOC0CTL.bit.CHSEL = 0;      // set SOC0 channel select to
ADCINA0/ADCINB0 pair
    Adc1Regs.ADCSOC0CTL.bit.TRIGSEL = 5;      // Set SOC0 start trigger to ADC
Trigger 1(EPWM1 SOCA) of the adc
    Adc1Regs.ADCSOC0CTL.bit.ACQPS = 6;       // set SOC0 S/H Window to 7 ADC Clock
Cycles, (6 ACQPS plus 1)

    Adc1Regs.ADCSOC2CTL.bit.CHSEL = 2;      // set SOC2 channel select to
ADCINA2/ADCINB2 pair
    Adc1Regs.ADCSOC2CTL.bit.TRIGSEL = 5;      // set SOC2 start trigger to ADC
Trigger 1(EPWM1 SOCA) of the adc
    Adc1Regs.ADCSOC2CTL.bit.ACQPS = 6;       // set SOC2 S/H Window to 7 ADC Clock
Cycles, (6 ACQPS plus 1)

    Adc1Regs.ADCSOC4CTL.bit.CHSEL = 3;    // set SOC4 channel select to
ADCINA3/ADCINB3 pair
    Adc1Regs.ADCSOC4CTL.bit.TRIGSEL = 5;      // set SOC4 start trigger to ADC
Trigger 1(EPWM1 SOCA) of the adc
    Adc1Regs.ADCSOC4CTL.bit.ACQPS = 6;       // set SOC4 S/H Window to 7 ADC Clock
Cycles, (6 ACQPS plus 1)

    Adc1Regs.ADCSOC6CTL.bit.CHSEL = 4;    // set SOC6 channel select to
ADCINA4/ADCINB4 pair
    Adc1Regs.ADCSOC6CTL.bit.TRIGSEL = 5;      // set SOC6 start trigger to ADC
Trigger 1(EPWM1 SOCA) of the adc
    Adc1Regs.ADCSOC6CTL.bit.ACQPS = 6;       // set SOC6 S/H Window to 7 ADC Clock
Cycles, (6 ACQPS plus 1)

    Adc1Regs.ADCSOC9CTL.bit.CHSEL = 6;    // set SOC8 channel select to
ADCINA6/ADCINB6 pair
    Adc1Regs.ADCSOC9CTL.bit.TRIGSEL = 5;      // set SOC8 start trigger to ADC
Trigger 1(EPWM1 SOCA) of the adc
    Adc1Regs.ADCSOC9CTL.bit.ACQPS = 6;       // set SOC8 S/H Window to 7 ADC Clock
Cycles, (6 ACQPS plus 1)

    Adc1Regs.ADCSOC11CTL.bit.CHSEL = 7;    // set SOC10 channel select to
ADCINA6/ADCINB6 pair
    Adc1Regs.ADCSOC11CTL.bit.TRIGSEL = 5;      // set SOC10 start trigger to ADC
Trigger 1(EPWM1 SOCA) of the adc
```

```
    Adc1Regs.ADCSOC11CTL.bit.ACQPS = 6;       // set SOC10 S/H Window to 7 ADC
Clock Cycles, (6 ACQPS plus 1)
    //OUTRO ADC 1- vanessa

    Adc1Regs.ADCSOC13CTL.bit.CHSEL = 8;    // set SOC8 channel select to
ADCINA6/ADCINB6 pair
    Adc1Regs.ADCSOC13CTL.bit.TRIGSEL = 5;      // set SOC8 start trigger to ADC
Trigger 1(EPWM1 SOCA) of the adc
    Adc1Regs.ADCSOC13CTL.bit.ACQPS = 6;       // set SOC8 S/H Window to 7 ADC
Clock Cycles, (6 ACQPS plus 1)

    Adc1Regs.ADCSOC15CTL.bit.CHSEL = 9;    // set SOC10 channel select to
ADCINA6/ADCINB6 pair
    Adc1Regs.ADCSOC15CTL.bit.TRIGSEL = 5;      // set SOC10 start trigger to ADC
Trigger 1(EPWM1 SOCA) of the adc
    Adc1Regs.ADCSOC15CTL.bit.ACQPS = 6;       // set SOC10 S/H Window to 7 ADC
Clock Cycles, (6 ACQPS plus 1)
    //OUTRO ADC 1- fim


    //Adc2-vanessa
      Adc2Regs.ADCSOC0CTL.bit.CHSEL = 0;      // set SOC0 channel select to
ADCINA0/ADCINB0 pair
        Adc2Regs.ADCSOC0CTL.bit.TRIGSEL = 5;      // Set SOC0 start trigger to ADC
Trigger 1(EPWM1 SOCA) of the adc
        Adc2Regs.ADCSOC0CTL.bit.ACQPS = 6;       // set SOC0 S/H Window to 7 ADC
Clock Cycles, (6 ACQPS plus 1)

        Adc2Regs.ADCSOC2CTL.bit.CHSEL = 2;      // set SOC2 channel select to
ADCINA2/ADCINB2 pair
        Adc2Regs.ADCSOC2CTL.bit.TRIGSEL = 5;      // set SOC2 start trigger to ADC
Trigger 1(EPWM1 SOCA) of the adc
        Adc2Regs.ADCSOC2CTL.bit.ACQPS = 6;       // set SOC2 S/H Window to 7 ADC
Clock Cycles, (6 ACQPS plus 1)

        Adc2Regs.ADCSOC4CTL.bit.CHSEL = 3;    // set SOC4 channel select to
ADCINA3/ADCINB3 pair
        Adc2Regs.ADCSOC4CTL.bit.TRIGSEL = 5;      // set SOC4 start trigger to ADC
Trigger 1(EPWM1 SOCA) of the adc
        Adc2Regs.ADCSOC4CTL.bit.ACQPS = 6;       // set SOC4 S/H Window to 7 ADC
Clock Cycles, (6 ACQPS plus 1)

        Adc2Regs.ADCSOC6CTL.bit.CHSEL = 4;    // set SOC6 channel select to
ADCINA4/ADCINB4 pair
        Adc2Regs.ADCSOC6CTL.bit.TRIGSEL = 5;      // set SOC6 start trigger to ADC
Trigger 1(EPWM1 SOCA) of the adc
        Adc2Regs.ADCSOC6CTL.bit.ACQPS = 6;       // set SOC6 S/H Window to 7 ADC
Clock Cycles, (6 ACQPS plus 1)

        Adc2Regs.ADCSOC9CTL.bit.CHSEL = 6;    // set SOC8 channel select to
ADCINA6/ADCINB6 pair
        Adc2Regs.ADCSOC9CTL.bit.TRIGSEL = 5;      // set SOC8 start trigger to ADC
Trigger 1(EPWM1 SOCA) of the adc
        Adc2Regs.ADCSOC9CTL.bit.ACQPS = 6;       // set SOC8 S/H Window to 7 ADC
Clock Cycles, (6 ACQPS plus 1)

        Adc2Regs.ADCSOC11CTL.bit.CHSEL = 7;    // set SOC10 channel select to
ADCINA6/ADCINB6 pair
```

```
        Adc2Regs.ADCSOC11CTL.bit.TRIGSEL = 5;      // set SOC10 start trigger to
ADC Trigger 1(EPWM1 SOCA) of the adc
        Adc2Regs.ADCSOC11CTL.bit.ACQPS = 6;       // set SOC10 S/H Window to 7 ADC
Clock Cycles, (6 ACQPS plus 1

        //OUTRO ADC 2- vanessa

            Adc2Regs.ADCSOC13CTL.bit.CHSEL = 8;    // set SOC8 channel select to
ADCINA6/ADCINB6 pair
            Adc2Regs.ADCSOC13CTL.bit.TRIGSEL = 5;      // set SOC8 start trigger
to ADC Trigger 1(EPWM1 SOCA) of the adc
            Adc2Regs.ADCSOC13CTL.bit.ACQPS = 6;       // set SOC8 S/H Window to 7
ADC Clock Cycles, (6 ACQPS plus 1)

            Adc2Regs.ADCSOC15CTL.bit.CHSEL = 9;    // set SOC10 channel select to
ADCINA6/ADCINB6 pair
            Adc2Regs.ADCSOC15CTL.bit.TRIGSEL = 5;      // set SOC10 start trigger
to ADC Trigger 1(EPWM1 SOCA) of the adc
            Adc2Regs.ADCSOC15CTL.bit.ACQPS = 6;       // set SOC10 S/H Window to 7
ADC Clock Cycles, (6 ACQPS plus 1)
            //OUTRO ADC 2- fim


    //Adc2-fim

    EDIS;

    BIOS_start();    // does not return

}


// FUNÇÃO disparada pelo módulo PWM1 que realiza a leitura dos ADs e acionamento
do LED

Void adc_fxn(UArg arg)
{

             Ia_carga_off = ((float)(Adc1Result.ADCRESULT4));          // DB2
             Ib_carga_off = ((float)(Adc1Result.ADCRESULT6));          // DB2
             Ic_carga_off= ((float)(Adc1Result.ADCRESULT8));                 //
DB2
             Ia_off = ((float)(Adc1Result.ADCRESULT1));                      //
DB3
             Ib_off = ((float)(Adc1Result.ADCRESULT11));               // DB3
             Ic_off = ((float)(Adc1Result.ADCRESULT13));               // DB3
             Tensao_valinha_off = ((float)(Adc1Result.ADCRESULT15));   // DB4
             Tensao_vblinha_off = ((float)(Adc2Result.ADCRESULT8));          //
DB4


             Ia_carga = ((Ia_carga_off) - 1915)*0.00348; // placa de corrente
             Ib_carga = ((Ib_carga_off) -1908)*0.00348; // placa de corrente
             Ic_carga= ((Ic_carga_off) -1902)*0.00344; // placa de corrente
             Tensao_valinha = ((Tensao_valinha_off) - 1874 )*0.601;  //0.63
             Tensao_vblinha = - ((Tensao_vblinha_off) - 1885 )*0.601;
             Tensao_vclinha = (- Tensao_valinha - Tensao_vblinha); //
```

```
            Tensao_va_fase = ((2*Tensao_valinha) + Tensao_vblinha)*0.333333; //
vafase=(1/3)*((2*vaborig)+ vbcorig);
            Tensao_vb_fase = (0.333333)*(- Tensao_valinha + Tensao_vblinha); //
vbfase=(1/3)*(-vaborig + vbcorig);
            Tensao_vc_fase = ( - Tensao_va_fase - Tensao_vb_fase);       //
Tensao de fase vac= -(vab+vbc)
            //Tensao_vclinha = (Tensao_vac - Tensao_vbc);
            Pot_ativa = (Tensao_va_fase *(Ia_carga)) + (Tensao_vb_fase *
(Ib_carga)) + (Tensao_vc_fase*(Ic_carga )); //+ (Tensao_vb_fase * (- Ib_carga)) +
;
            Pot_reativa = (((Tensao_va_fase - Tensao_vb_fase) * Ic_carga) +
((Tensao_vb_fase - Tensao_vc_fase) * Ia_carga) + ((Tensao_vc_fase -
Tensao_va_fase) * Ib_carga))*0.5773502;
            Pot_aparente = sqrt(((Pot_ativa)*(Pot_ativa)) +
((Pot_reativa)*(Pot_reativa)));
            Fat_potencia = (Pot_ativa) / (Pot_aparente);


                  //-------------------------------------------------------------
                  // Filtro de média móvel
                  //-------------------------------------------------------------

                  filtro_ativo[z] = (int) Pot_ativa;
                  filtro_reativo[z]= (int) Pot_reativa;
                  filtro_aparente[z] = (int) Pot_aparente;
                  z++;
                  if (z==200) {
                        z=0;
                                    }

                  for (j=0; j<200; j++){
                        Pot_ativa_aux += filtro_ativo[j];
                        Pot_reativa_aux += filtro_reativo[j];
                        Pot_aparente_aux += filtro_aparente[j];
                  }

                        Pot_ativa_filt = (Pot_ativa_aux * 0.005);
                        Pot_reativa_filt = (Pot_reativa_aux * 0.005);
                        Pot_aparente_filt = (Pot_aparente_aux * 0.005);



                        Pot_ativa_aux = 0;
                        Pot_reativa_aux =0;
                        Pot_aparente_aux =0;

                  //-------------------------------------------------------------
--------
                  //-------------------------------------------------------------

            //transformando para inteiro *100 para enviar para o nucleo de
comunicação
            Tensao_va_fase_i = (long int) (Tensao_va_fase*100);
            Tensao_vb_fase_i = (long int) (Tensao_vb_fase*100);
            Tensao_vc_fase_i = (long int) (Tensao_vc_fase*100);
            Tensao_valinha_i = (long int) (Tensao_valinha*100);
            Tensao_vblinha_i = (long int) (Tensao_vblinha*100);
            Tensao_vclinha_i = (long int) (Tensao_vclinha*100);
```

```
            Ia_carga_i = (long int) (Ia_carga*100);
            Ib_carga_i = (long int) (Ib_carga*100);
            Ic_carga_i = (long int) (Ic_carga*100);
            Pot_ativa_i = (long int) (Pot_ativa_filt*100);
            Pot_reativa_i = (long int) (Pot_reativa_filt*100);
            Pot_aparente_i = (long int) (Pot_aparente_filt*100);
            Ia_i = (long int) (Ia*100);
            Ib_i = (long int) (Ib*100);
            Ic_i = (long int) (Ic*100);
            Fat_potencia_i = (long int) (Fat_potencia*100);


                                buffer01[k]  =  (int)(Tensao_valinha); // leitura
placa de tensao
            //buffer01[k] = medida_01;
                               buffer02[k]  =  (int)(Tensao_vblinha); //leitura placa
de tensao
                               buffer03[k]  =  (int)(Ia_carga*100); //leitura placa de
corrente
                               buffer04[k]  =  (int)(Ib_carga); //leitura placa de
corrente
                               buffer05[k]  =  (int)(Ic_carga*100); //leitura placa de
corrente
                               buffer06[k]  =  (int)(Pot_ativa_filt);
                               buffer07[k]  =  (int)(Pot_reativa_filt);
                               buffer08[k]  =  (int)(Pot_aparente_filt);

                               // buffer08[k] = (Ib_carga);



                        //-- Contador usado para salvar buffers
                           if(k == BUFFER_SIZE1-1)   {
                                 k = 0 ;

                                                    // Circular buffer
                           }
                           else k++;

        count_led++; // temporizador de acionamento do LED
        if (count_led < 5000) GpioG1DataRegs.GPADAT.bit.GPIO31 = 0;
        if (count_led > 5000 && count_led < 10000)
GpioG1DataRegs.GPADAT.bit.GPIO31 = 1;
        if (count_led > 10000 ) count_led = 0;


         interrupcao ++; // variável de controle

         Adc1Regs.ADCINTFLGCLR.bit.ADCINT1 = 1;  //Clear ADCINT1 flag reinitialize
                                                 // for next SOC

}

//TODO
```

APÊNDICE B

Desenvolvimento da aplicação para o núcleo Cortex-M3, que é responsável pela comunicação com o núcleo C28x e pela comunicação com o sistema de supervisão via *Modbus* TCP/IP.

```
//////////////////////////////////////////////////////////////////////////////
////////
// UNIVERSIDADE FEDERAL DE SANTA MARIA-UFSM
// GRUPO DE ELETRONICA DE POTENCIA E CONTROLE-GEPOC
//////////////////////////////////////////////////////////////////////////////
///////
// Código referente ao núcleo Cortex-M3 do DSP Concerto F28M36x
// AUTORA: Vanessa Furtado de Lima
//////////////////////////////////////////////////////////////////////////////
//////
// DESCRIÇÃO:
// Este código realiza:
// > Comunicação com núcleo C28x via MESSAGEQ
// > Comunicação com Computador via Modbus TCP/IP
//////////////////////////////////////////////////////////////////////////////
/////
//CODE

/* XDCtools Header files */
#include <xdc/std.h>
#include <xdc/cfg/global.h>
#include <xdc/runtime/Diags.h>
#include <xdc/runtime/Error.h>
#include <xdc/runtime/IHeap.h>
#include <xdc/runtime/Log.h>
#include <xdc/runtime/Memory.h>
#include <xdc/runtime/System.h>

/* IPC Header files */
#include <ti/ipc/MessageQ.h>
#include <ti/ipc/MultiProc.h>

/* BIOS Header files */
#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/heaps/HeapBuf.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Clock.h>


/* TI-RTOS Header files */
#include <ti/drivers/SDSPI.h>
#include <ti/drivers/I2C.h>
#include <ti/drivers/GPIO.h>


 /* NDK Header files */
#include <ti/ndk/inc/netmain.h>
#include <ti/ndk/inc/_stack.h>
```

```
/* Example/Board Header file */
#include "TMDXDOCK28M36.h"

#include "demo.h"
#include "USBCDCD_LoggerIdle.h"

#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <file.h>

//setup M3

#include "inc/hw_sysctl.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_nvic.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "board_drivers/set_pinout_f28m36x.h"
#include "driverlib/ipc.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/debug.h"
#include "driverlib/cpu.c"
#include "driverlib/gpio.h"

#define TCPPACKETSIZE 1024  // Tamanho pacote TCP IP, 1024 BYTES
#define TCPPORT 502 // Porta TCP:1000
#define NUMTCPWORKERS 3

extern void SerialInit();
extern void SerialMain();
long int P_grid_i;
long int Q_grid_i;
long int V_Sgrid_i;

// Variáveis Supervisão MCH
 long int Tensao_va_fase_i , Tensao_vb_fase_i, Tensao_vc_fase_i, Tensao_valinha_i,
Tensao_vblinha_i, Tensao_vclinha_i, Ia_carga_i, Ib_carga_i, Ic_carga_i,
 Pot_ativa_i, Pot_reativa_i, Pot_aparente_i, Ia_i, Ib_i, Ic_i, Fat_potencia_i;

//////////////////////Modbus/////////////////////////////////////////////////////
// Global variables
#define Bilhete      30       // Tamanho máximo dos Bilhetes
#define BufSize      420       // Tamanho do Buffer do DSP = número de
registradores * 2 (L + H)

Uint8 BilheteModbusR[Bilhete]; //Tabela para leitura do Bilhete Modbus
Uint8 BilheteModbusT[Bilhete]; //Tabela para transmissão do Bilhete Modbus
Uint16 BufferModbus[BufSize];   //Tabela com valores das variáveis no DSP
Uint16 BilheteModbusR2[Bilhete];
long int nbytes2;
long int nbytes3;

Uint16 EnderecoModbus = 1; //Endereço do ESCRAVO (Protocolo Modbus)

Uint16 i;
Uint16 iR = 0;       // Indice do bilhete recebido
```

```
Uint16 iT;       // Indice do bilhete enviado
Uint16 EndInic;  // Endereço Inicial Bilhete Modbus
Uint16 NumDados; // Numero de Bytes Bilhete Modbus

Uint16 CodErro = 0;
Uint16 auxva = 15;

int BUFFER_SIZE1=1024;
int buffermodbus_teste1[1024], buffermodbus_teste2[1024];
int k;
////////////////////////////////////////////////////////////////////////////

int recordingEnabled = RECORDING_CLOSED, l =0;
char statusDSP = 'N', comms = 'D', FLAG_BUF0;

//Função que realiza comunicação com MCU F28335 via MESSAGEQ
/*
 *  ======== temperature_func ========
 *  Allocates a message and ping-pongs the message around the processors.
 *  A local message queue is created and a remote message queue is opened.
 *  Messages are sent to the remote message queue and retrieved from the
 *  local MessageQ.
 */
Void temperature_func(UArg arg0, UArg arg1)
{
    MessageQ_Msg     msg;
    MessageQ_Handle  messageQ;
    MessageQ_QueueId remoteQueueId;
    int              status;
    Ptr              buf;
    HeapBuf_Handle   heapHandle;
    HeapBuf_Params   hbparams;
    SizeT            blockSize;
    unsigned int     numBlocks;
    //unsigned int   i = 0;
    FILE            *dst;
    char             logBuffer[40];
    //SDSPI_Handle   sdspiHandle;
    Error_Block      eb;

    /* Compute the blockSize & numBlocks for the HeapBuf */
    numBlocks = 16; // Número de Blocos da árvore binária (buffer do MessageQ)
deve ser múltiplo de 8 (2^n)
    blockSize = sizeof(TempMsg);

    /* Alloc a buffer from the default heap */
    buf = Memory_alloc(0, numBlocks * blockSize, 0, NULL);

    /*
     *  Create the heap that is used for allocating MessageQ messages.
     */
    Error_init(&eb);
    HeapBuf_Params_init(&hbparams);
    hbparams.align          = 0;
    hbparams.numBlocks      = numBlocks;
    hbparams.blockSize      = blockSize;
    hbparams.bufSize        = numBlocks * blockSize;
    hbparams.buf            = buf;
    heapHandle = HeapBuf_create(&hbparams, &eb);
```

```
if (heapHandle == NULL) {
    System_abort("HeapBuf_create failed\n" );
}

/* Register default system heap with MessageQ */
MessageQ_registerHeap((IHeap_Handle)(heapHandle), HEAPID);

/* Create the local message queue */
messageQ = MessageQ_create(M3QUEUENAME, NULL);
if (messageQ == NULL) {
    System_abort("MessageQ_create failed\n" );
}

/* Open the remote message queue. Spin until it is ready. */
do {
    status = MessageQ_open(C28QUEUENAME, &remoteQueueId);
    /*
     *  Sleep for 1 clock tick to avoid inundating remote processor
     *  with interrupts if open failed
     */
    if (status < 0) {
        Task_sleep(1);
    }
} while (status < 0);

/* Allocate a message to be ping-ponged around the processors */
msg = MessageQ_alloc(HEAPID, sizeof(TempMsg));
if (msg == NULL) {
   System_abort("MessageQ_alloc failed\n" );
}

MessageQ_setMsgId(msg, TEMPERATURE_CONVERSION);

/*
 *  Send the message to the remote processor and wait for a message
 *  from the previous processor.
 */
System_printf("Start the main loop\n");
while (true) {

    Log_print0(Diags_USER1, "sending msg");
    /* send the message to the remote processor */
    status = MessageQ_put(remoteQueueId, msg);
    if (status < 0) {
       System_abort("MessageQ_put had a failure/error\n");
    }
    Log_print0(Diags_USER1, "getting msg");

    /* Get a message */
    status = MessageQ_get(messageQ, &msg, MessageQ_FOREVER);
    if (status < 0) {
       System_abort("This should not happen since timeout is forever\n");
    }
    Log_print0(Diags_USER1, "got msg");

    Tensao_va_fase_i = ((TempMsg *)msg)->Tensao_va_fase;
    Tensao_vb_fase_i =      ((TempMsg *)msg)->Tensao_vb_fase;
    Tensao_vc_fase_i = ((TempMsg *)msg)->Tensao_vc_fase;
    Tensao_valinha_i = ((TempMsg *)msg)->Tensao_valinha;
```

```
Tensao_vblinha_i = ((TempMsg *)msg)->Tensao_vblinha;
Tensao_vclinha_i = ((TempMsg *)msg)->Tensao_vclinha;
Ia_carga_i = ((TempMsg *)msg)->Ia_carga;
Ib_carga_i = ((TempMsg *)msg)->Ib_carga;
Ic_carga_i = ((TempMsg *)msg)->Ic_carga;
Pot_ativa_i = ((TempMsg *)msg)->Pot_ativa_filt;
Pot_reativa_i = ((TempMsg *)msg)->Pot_reativa_filt;
Pot_aparente_i = ((TempMsg *)msg)->Pot_aparente_filt;
Ia_i = ((TempMsg *)msg)->Ia;
Ib_i = ((TempMsg *)msg)->Ib;
Ic_i = ((TempMsg *)msg)->Ic;
Fat_potencia_i = ((TempMsg *)msg)->Fat_potencia;


BufferModbus[0] = 0; // Parte alta
BufferModbus[1] = 1; // Parte baixa


   BufferModbus[2] = (Tensao_valinha_i >> 8); // Parte alta
   BufferModbus[3] = (Tensao_valinha_i & 0xFF); // Parte baixa
   BufferModbus[4] = (Tensao_vblinha_i >> 8);
   BufferModbus[5] = (Tensao_vblinha_i & 0xFF);
   BufferModbus[6] = (Tensao_vclinha_i >> 8);
   BufferModbus[7] = (Tensao_vclinha_i & 0xFF);

   BufferModbus[8] = (2000 >> 8);  // Parte alta teste
   BufferModbus[9] = (2000 & 0xFF); // Parte baixa teste

   BufferModbus[10] = (Tensao_va_fase_i >> 8);
   BufferModbus[11]= (Tensao_va_fase_i & 0xFF);
   BufferModbus[12] = (Tensao_vb_fase_i >> 8);
   BufferModbus[13]= (Tensao_vb_fase_i & 0xFF);
   BufferModbus[14] = (Tensao_vc_fase_i >> 8);
   BufferModbus[15]= (Tensao_vc_fase_i & 0xFF);

   BufferModbus[16] = (Ia_carga_i >> 8);
   BufferModbus[17] = (Ia_carga_i & 0xFF);
   BufferModbus[18] = (Ib_carga_i >> 8);
   BufferModbus[19] = (Ib_carga_i & 0xFF);
   BufferModbus[20] = (Ic_carga_i >> 8);
   BufferModbus[21] = (Ic_carga_i & 0xFF);

   BufferModbus[22] = (Pot_ativa_i >> 8);
   BufferModbus[23] = (Pot_ativa_i & 0xFF);
   BufferModbus[24] = (Pot_reativa_i >> 8);
   BufferModbus[25] = (Pot_reativa_i & 0xFF);
   BufferModbus[26] = (Pot_aparente_i >> 8);
   BufferModbus[27] = (Pot_aparente_i & 0xFF);
   BufferModbus[28] = (Fat_potencia_i >> 8);
   BufferModbus[29] = (Fat_potencia_i & 0xFF);

   BufferModbus[30] = (Ia_i >> 8);
   BufferModbus[31] = (Ia_i & 0xFF);
   BufferModbus[32] = (Ib_i >> 8);
   BufferModbus[33] = (Ib_i & 0xFF);
   BufferModbus[34] = (Ic_i >> 8);
   BufferModbus[35] = (Ic_i & 0xFF);
```

```
        if (recordingEnabled == RECORDING_WRITING) {
        Log_print0(Diags_USER1, "writing SD card");
        /* Format the logBuffer to a writeable char string */
        // System_sprintf(logBuffer, "%u,%d\n", i++, (int)temperatureC);

            /* Write buffer to the SD Card */
            fwrite(logBuffer, 1, strlen(logBuffer), dst);
            fflush(dst);
            Log_print0(Diags_USER1, "done with SD card");
        }

        Log_print0(Diags_USER1, "sleep");
        Task_sleep(200); // SLEEP por 200 clock ticks
        Log_print0(Diags_USER1, "awake");
    }
}


//Rotina que gerencia conexão TCP
/*
 *  ======== tcpWorker ========
 *  Task to handle TCP connection. Can be multiple Tasks running
 *  this function.
 */
Void tcpWorker(UArg arg0, UArg arg1)
{
    SOCKET clientfd = (SOCKET)arg0;
    int nbytes;
    bool flag = true;
    char *buffer;
    int nbytesmodbus;


    Error_Block eb;


    fdOpenSession(TaskSelf());

    System_printf("tcpWorker: start clientfd = 0x%x\n", clientfd);

    /* Make sure Error_Block is initialized */
    Error_init(&eb);

    /* Get a buffer to receive incoming packets. Use the default heap. */
    buffer = Memory_alloc(NULL, TCPPACKETSIZE, 0, &eb);
    if (buffer == NULL) {
        System_printf("tcpWorker: failed to alloc memory\n");
        Task_exit();
    }



    /* Loop while we receive data */
    while (flag) {
      //        nbytes = recv(clientfd, (char *)buffer, TCPPACKETSIZE, 0);

      nbytes = recv(clientfd, BilheteModbusR, TCPPACKETSIZE, 0);
```

```
      nbytes2 = (BilheteModbusR[0]); // variáveis auxiliares para receber o id de
transação
      nbytes3 = (BilheteModbusR[1]); // de valor maior que 256


        //System_printf(" %c \n", (char *)buffer[0]);

        // ----- MENSAGEM PARA MIM ---------------------------------------
         if (BilheteModbusR[6] == EnderecoModbus) // Se mensagem é para este
endereço (primeiro byte)
         {   auxva=22;

            BilheteModbusT[6] = BilheteModbusR[6]; // Endereço do bilhete a ser
enviado = recebido
            BilheteModbusT[7] = BilheteModbusR[7]; // Código Função do bilhete a
ser enviado = recebido
                     if (nbytes3 < 256) {
            BilheteModbusT[1] = BilheteModbusR[1]; // Identificação de transação a
ser enviado = recebido
            BilheteModbusT[0] = 0;

                                        }
                     if (nbytes2 > 0) {                          //Cado o
identificador de transação seja de valor maior que 255
                           BilheteModbusT[0] = (nbytes2);  //nbytes2 =
(BilheteModbusR[0])
                           BilheteModbusT[1]= (nbytes3);   //nbytes3 =
(BilheteModbusR[1])
                                                   }

            // Testa se endereço é válido
            EndInic = (BilheteModbusR[8]<<8 & 0xFF00) + BilheteModbusR[9];
              if (EndInic > (BufSize - 1)) // Este endereço varia de acordo com o
tamanho do Buffer de registradores definido
            {
               CodErro = 2; // Endereço inválido
               //goto ERRO;
            }

            switch (BilheteModbusR[7]){ // O Segundo byte é relativo a função
MODBUS, que pode ser:

            //----- FUNÇÃO 03 = LEITURA --------------------------------
            case 3: // Se BilheteModbusR [7] = 3
               NumDados = ((BilheteModbusR[10]<<8 & 0xFF00) +
(BilheteModbusR[11])*2); // Número de bytes a serem lidos
               iT = 8; // variação do índice do bilhete a ser transmitido. Começa
no (3 + iT++) 4 pois os 3 primeiros
                        // são 1) endereço, 2) função e 3) número de bytes

               // NumBytes = Número de registradores a ser lidos compostos de
parte alta e parte baixa
               // Multiplica por 2 por ser dividido em 2 partes
               auxva=10;

               for (i=EndInic; i < (EndInic + (NumDados)); i++)
                  {
                     iT++;
                     auxva=12;
```

```
                    BilheteModbusT[iT] = BufferModbus[i];
                }

                BilheteModbusT[8] = NumDados;   // O número de dados (byte alto
+ byte baixo = 1 dado)
                BilheteModbusT[5] = (NumDados)+3;   // O número de dados (byte
alto + byte baixo = 1 dado)


            break;

            //----- FUNÇÃO 06 = ESCREVE 1 REGISTRADOR -------------------
               case 06:
                    BufferModbus[EndInic] = BilheteModbusR[10];     // Escreve
parte alta
                    BufferModbus[EndInic + 1] = BilheteModbusR[11]; // Escreve
parte baixa
                    auxva=50;
                    for (iT=8; iT<= 11; iT++)
                    {
                       BilheteModbusT[iT] = BilheteModbusR[iT];  // O bilhete a
ser transmitido é igual ao recebido
                    }
                    iT = 11;
                     break;


                    //----- FUNÇÃO 16 = ESCREVE VÁRIOS REGISTRADORES -----------
-
                     case 16:
                                NumDados = ((BilheteModbusR[10]<<8 & 0xFF00) +
(BilheteModbusR[11])*2); // Número de bytes a serem escritos

                                     iT = BilheteModbusR[12]; // Número de
bytes de dados

                                iT = iT + 12;
                               BilheteModbusT[5] = (NumDados)+4;   // O número de
dados (byte alto + byte baixo = 1 dado)

                              for (i=13; i<=iT; i++)
                                {
                                   BufferModbus[EndInic] = BilheteModbusR[i]; //
Escreve no Buffer do DSP os dados do Bilhete
                                     EndInic++;
                                }

                              for (i=8; i<= 14; i++)
                                {
                                   BilheteModbusT[i] = BilheteModbusR[i];  // O
bilhete a ser transmitido é igual ao recebido
                                }                                          // até
o byte 6
                                iT = 14;
                               break;

          }
        }
```

```
          // nbytesmodbus = recv(clientfd, BilheteModbusT, TCPPACKETSIZE, 0);
        if (nbytes > 0) {

                        auxva=14;
                    nbytesmodbus = (iT)+1;

                    send(clientfd, BilheteModbusT, nbytesmodbus, 0);

                    //Teste buffer modbus com leitura de sinais analogicos da placa
de corrente
                    buffermodbus_teste1[k]  = (int)(P_grid_i >> 8);
                    buffermodbus_teste2[k]  = (int)(P_grid_i & 0xFF);

                    //  buffer01[k]  =  (int)(Q_grid);
                    //                   buffer02[k]  =  (int)(P_grid);
                                            if(k == BUFFER_SIZE1-1)   k = 0;
                    // Circular buffer
                                            else k++;
                                            //final teste

}
        else {
            fdClose(clientfd);
            flag = false;
        }
    }
    System_printf("tcpWorker stop clientfd = 0x%x\n", clientfd);

    /* Free the buffer back to the heap */
    Memory_free(NULL, buffer, TCPPACKETSIZE);

    fdCloseSession(TaskSelf());
    /*
     *  Since deleteTerminatedTasks is set in the cfg file,
     *  the Task will be deleted when the idle task runs.
     */
    Task_exit();
}

//Rotina que incia e cria conexões TCP,  sockets
/*
 *  ======== tcpHandler ========
 *  Creates new Task to handle new TCP connections.
 */
Void tcpHandler(UArg arg0, UArg arg1)
{
    SOCKET lSocket;
    struct sockaddr_in sLocalAddr;
    SOCKET clientfd;
    struct sockaddr_in client_addr;
    int addrlen=sizeof(client_addr);
    int optval;
    int optlen = sizeof(optval);
    int status;
    Task_Handle taskHandle;
    Task_Params taskParams;
    Error_Block eb;

    fdOpenSession(TaskSelf());
```

```
    lSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (lSocket < 0) {
        System_printf("tcpHandler: socket failed\n");
        Task_exit();
        return;
    }

    memset((char *)&sLocalAddr, 0, sizeof(sLocalAddr));
    sLocalAddr.sin_family = AF_INET;
    sLocalAddr.sin_len = sizeof(sLocalAddr);
    sLocalAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    sLocalAddr.sin_port = htons(arg0);

    status = bind(lSocket, (struct sockaddr *)&sLocalAddr, sizeof(sLocalAddr));
    if (status < 0) {
        System_printf("tcpHandler: bind failed\n");
        fdClose(lSocket);
        Task_exit();
        return;
    }

    if (listen(lSocket, NUMTCPWORKERS) != 0){
        System_printf("tcpHandler: listen failed\n");
        fdClose(lSocket);
        Task_exit();
        return;
    }

    if (setsockopt(lSocket, SOL_SOCKET, SO_KEEPALIVE, &optval, optlen) < 0) {
        System_printf("tcpHandler: setsockopt failed\n");
        fdClose(lSocket);
        Task_exit();
        return;
    }

    while (true) {
        /* Wait for incoming request */
        clientfd = accept(lSocket, (struct sockaddr*)&client_addr, &addrlen);
        System_printf("tcpHandler: Creating thread clientfd = %d\n", clientfd);

        /* Init the Error_Block */
        Error_init(&eb);

        /* Initialize the defaults and set the parameters. */
        Task_Params_init(&taskParams);
        taskParams.arg0 = (UArg)clientfd;
        taskParams.stackSize = 1024;
        taskHandle = Task_create((Task_FuncPtr)tcpWorker, &taskParams, &eb);
        if (taskHandle == NULL) {
            System_printf("tcpHandler: Failed to create new Task\n");
        }
    }
}

// Código main, incializa módulos GPIO, UART, ETHERNET-MAC e LIBERA PORTAS GPIO
int main(void)
{
    Task_Handle taskHandle;
```

```
    Task_Params taskParams;
    Error_Block eb;

    /* Set up the board specific items */
    TMDXDOCK28M36_initGeneral();
    TMDXDOCK28M36_initUART();
    TMDXDOCK28M36_initGPIO();
    TMDXDOCK28M36_initSDSPI();
    TMDXDOCK28M36_initUSB(TMDXDOCK28M36_USBDEVICE);
    TMDXDOCK28M36_initEMAC();

    System_printf("Demo with HTTP, I2C, and SD\nSystem provider is set to SysMin,
halt the target and use ROV to view output.\n");
    /* SysMin will only print to the console when you call flush or exit */
    System_flush();

    /* Liga LED 2 do kit Concerto */
    //GPIO_write(TMDXDOCK28M36_D1, TMDXDOCK28M36_LED_ON);

    USBCDCD_init();
    SerialInit();
    InitBuffers(); // Inicia buffers interno DSP, e bilhetes ModBus


    /*
    *  Create the Task that farms out incoming TCP connections.
    *  arg0 will be the port that this task listens to.
    */
    Task_Params_init(&taskParams);
    Error_init(&eb);

    taskParams.stackSize = 1024;
    taskParams.priority = 1;
    taskParams.arg0 = TCPPORT;
    taskHandle = Task_create((Task_FuncPtr)tcpHandler, &taskParams, &eb);
    if (taskHandle == NULL) {
        System_printf("main: Failed to create tcpHandler Task\n");
    }

    /* Create the Task that communicates to I2C temperature sensor. */
    Task_Params_init(&taskParams);
    taskParams.stackSize = 1280;
    taskParams.priority = 10;
    taskParams.instance->name = "Temperature";
    taskHandle = Task_create(temperature_func, &taskParams, &eb);
    if (taskHandle == NULL) {
        System_printf("Task_create() failed!\n");
        BIOS_exit(0);
    }
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE); //funcionou
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOH);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);
    //SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOK); Prejudica o Ethernet
```

```
    //SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOL);
    //SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOM); Prejudica o Ethernet
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOP);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOQ);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOR);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOS);

    GPIOPadConfigSet(GPIO_PORTA_BASE, 0xFF, GPIO_PIN_TYPE_STD_WPU);
    GPIOPadConfigSet(GPIO_PORTB_BASE, 0xFF, GPIO_PIN_TYPE_STD_WPU);

    // Give C28 control of all GPIOs
    GPIOPinConfigureCoreSelect(GPIO_PORTA_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
    GPIOPinConfigureCoreSelect(GPIO_PORTB_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
    GPIOPinConfigureCoreSelect(GPIO_PORTC_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
    GPIOPinConfigureCoreSelect(GPIO_PORTD_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
    GPIOPinConfigureCoreSelect(GPIO_PORTE_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
    GPIOPinConfigureCoreSelect(GPIO_PORTF_BASE, 0xFF, GPIO_PIN_M_CORE_SELECT);
//Nucleo M3 controla GPIO's da porta F
    GPIOPinConfigureCoreSelect(GPIO_PORTG_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
    GPIOPinConfigureCoreSelect(GPIO_PORTH_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
    GPIOPinConfigureCoreSelect(GPIO_PORTJ_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
    //GPIOPinConfigureCoreSelect(GPIO_PORTK_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
Prejudica o Ethernet
    //GPIOPinConfigureCoreSelect(GPIO_PORTL_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
    //GPIOPinConfigureCoreSelect(GPIO_PORTM_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
Prejudica o Ethernet
    GPIOPinConfigureCoreSelect(GPIO_PORTN_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
    GPIOPinConfigureCoreSelect(GPIO_PORTP_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
    GPIOPinConfigureCoreSelect(GPIO_PORTQ_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
    GPIOPinConfigureCoreSelect(GPIO_PORTR_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
    GPIOPinConfigureCoreSelect(GPIO_PORTS_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);

     /*Start BIOS. Will not return from this call. */


    statusDSP = 'O';
    BIOS_start();

    return (0);
}


/*função modbus para inicializar o buffer */
void InitBuffers(void)
{
    //Inicializa BUFFERS

   for(i=0 ; i<Bilhete ; i++)
   {
      BilheteModbusR[i] = 0;
        BilheteModbusT[i] = 0;
        auxva=3;
   }
   i = 0;
   iR = 0;


}
```