UNIVERSIDADE DE CAXIAS DO SUL – UCS
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA – CCET
CIDADE UNIVERSITÁRIA
ENGENHARIA DE CONTROLE E AUTOMAÇÃO

LEANDRO POHLMANN ROCHA

SISTEMA EMBARCADO DE COMUNICAÇÃO EM AMBIENTE CORPORATIVO DE FÁBRICA

CAXIAS DO SUL
2015

LEANDRO POHLMANN ROCHA

SISTEMA EMBARCADO DE COMUNICAÇÃO EM AMBIENTE CORPORATIVO DE FÁBRICA

Trabalho de conclusão de curso de graduação, apresentado ao Centro de Ciências Exatas da Natureza e de Tecnologia da Universidade de Caxias do Sul, como requisito para a obtenção do grau de Engenheiro de Controle e Automação.

Orientador: Prof. Dr. Stanislav Tairov

CAXIAS DO SUL

2015

LEANDRO POHLMANN ROCHA

SISTEMA EMBARCADO DE COMUNICAÇÃO EM AMBIENTE CORPORATIVO DE FÁBRICA

Trabalho de conclusão de curso de graduação, apresentado ao Centro de Ciências Exatas da Natureza e de Tecnologia da Universidade de Caxias do Sul, como requisito para a obtenção do grau de Engenheiro de Controle e Automação.

Aprovado em _____de _____ de _____

Banca Examinadora

_____
Professor
Universidade de Caxias do Sul – UCS

_____
Professor
Universidade de Caxias do Sul – UCS

_____
Professor
Universidade de Caxias do Sul – UCS

AGRADECIMENTOS

À Vida.

À minha família, base para todos os objetivos alcançados, agradeço pelo apoio e incentivo.

Ao meu orientador, Prof. Dr. Stanislav Tairov pelo conhecimento, dedicação e competência na missão de ensinar.

Aos meus amigos e colegas de curso que compartilharam momentos durante toda a graduação.

RESUMO

O trabalho proposto apresenta o desenvolvimento de um sistema embarcado para realizar a integração da informação das máquinas do chão de fábrica com o ambiente de gerência das empresas. Chamado de Interface Integradora, esse sistema possui um servidor embarcado, sendo capaz de adquirir os dados das máquinas ponto a ponto, processá-los e disponibilizá-los na rede local em que estiver conectado. A aquisição de dados foi realizada através de protocolo de comunicação Modbus, ou através de sinais digitais e sinais analógicos. O servidor embarcado conecta-se na rede local da organização através do protocolo Ethernet TCP/IP. A implementação deste sistema utilizou um microcontrolador ARM Cortex-M3, sendo o foco deste trabalho o desenvolvimento do software para a aplicação. Além disso, um software configurador baseado em PC foi desenvolvido na plataforma LabVIEW® da National Instruments, permitindo que o sistema seja simples de instalar, configurar e operar. Ao fim, será proposto a simulação de funcionamento e, posteriormente, testes interligando controladores industriais a uma rede local foram realizados para validar a solução.

Palavras-chave: Interligação chão de fábrica/organização, microcontroladores, comunicação de dados, Modbus.

ABSTRACT

This paper proposes the development of an embedded system for the integration of information between shop floor machines and corporate business environment. This system has a web server embedded, which is able to acquire data from machines, process and make it available to the local network.. The data acquisition can be performed either via Modbus communication protocol, digital signals or analog signals. The web server is connected to the local network of the organization via Ethernet TCP / IP protocol. The implementation of this system is based on a ARM Cortex-M3 microcontroller, focusing on the software's development for the application. In addition, a PC-based configuration software was developed in LabVIEW® from National Instruments, allowing simple installing, configuring and operating. The working simulation is proposed, and industrial controllers connected to the LAN were performed to validate the solution.

Keywords: Integration shop floor/enterprise, microcontrollers, data communication systems, Modbus.

SUMÁRIO

LISTA DE FIGURAS

GLOSSARIO

ADC	*Analog-Digital-Converter*
API	*Application Programming Interface*
ARM	*Advanced RISC Machine*
ASCII	*American Standard Code for Information Interchange*
CRC	*Cyclic Redundancy Check*
CNC	Comando Numérico Computadorizado
CLP	Controlador Lógico Programável
ERP	*Enterprise Resource Planning*
GPIO	*General Purpose Input/Output*
IDE	*Integrated Development Environment*
IHM	Interface Homem-Máquina
I/O	*Input/Output*
LSB	*Least Significant Byte*
MES	*Manufacturing Execution System*
MSB	*Most Significant Byte*
RAM	*Random Access Memory*
RTU	*Remote Terminal Unit*
SCADA	Supervisory Control and Data Acquisition
UART	*Universal Asynchronous Receiver and Transmitter*

1 INTRODUÇÃO

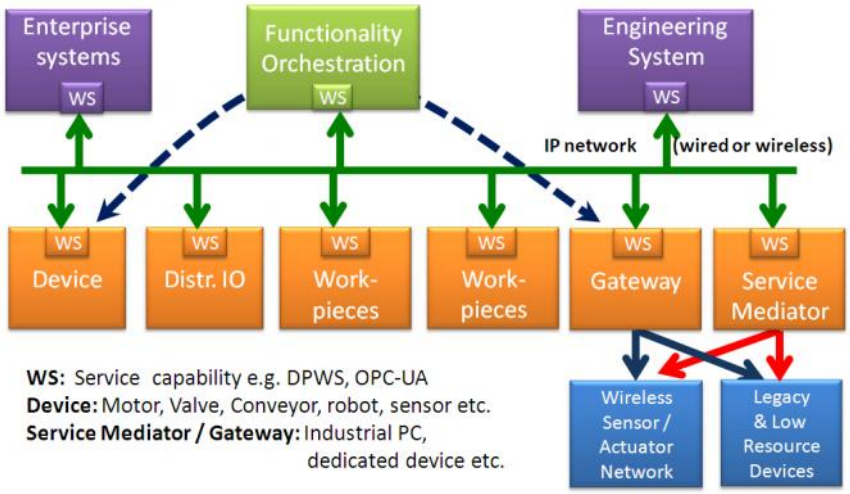
As máquinas e equipamentos industriais realizam operações essenciais para os processos produtivos das empresas. Porém, a realidade é caracterizada pela falta de interligação entre os níveis hierárquicos da organização, principalmente entre chão de fábrica e ambiente corporativo. Os gestores e tomadores de decisão, na maioria dos casos, não possuem acesso aos dados das máquinas em tempo real e as informações de produção geralmente são adquiridas e transmitidas aos administradores através de processos manuais. Entretanto, o mercado global exige que as empresas tenham uma reação rápida e eficiente a eventos críticos que ocorrem na sua linha de produção. Assim, todas as organizações que objetivam atingir altos níveis de qualidade, produtividade e redução de custos, precisam integrar os dados no nível de máquina com os processos de negócios.

Historicamente, de acordo com (ZURAWSKI, 2014). as tecnologias de redes de comunicação no ambiente corporativo e no ambiente produtivo evoluíram separadamente e tiveram de atender a diferentes exigências. O ambiente corporativo sempre esteve relacionado a estratégia, focado nos dados de gestão e em conceitos de rede utilizando predominantemente redes locais LAN. Já no ambiente de produção, os objetivos são a busca contínua pela excelência operacional, com foco em dados de controle em tempo real, sendo este nível de domínio, na grande maioria, atendido por barramentos seriais padronizados ou proprietários. Assim, percebe-se que a interligação entre os níveis organizacionais não diz respeito somente a ligação de dois tipos de rede diferentes, mas também a interligação de dois conceitos distintos.

Uma excelente abordagem para esta interligação vertical dos processos industriais é fazer com que o chão de fábrica trabalhe orientado a serviços como mostra (COLOMBO e KARNOUSKOS, 2011). A abordagem de Arquitetura Orientada a Serviços (SOA), é um paradigma que propõe interconexão dos dispositivos a uma rede comum a todos os equipamentos e computadores da organização (Figura 1-1). A filosofia é que cada componente ofereça determinadas funções e informações na forma de serviços aos demais componentes da rede, de acordo com seus interesses. Estas características trazem muitas vantagens como, a

intercomunicação das máquinas, flexibilização do layout da fábrica, acrescentar e excluir componentes da rede sem realizar mudanças de configuração de software, além de criar diversas aplicações a partir dos dados dos equipamentos que estão disponíveis na rede.

Figura 1-1 - Proposta de intercomunicação das máquinas e flexibilização do layout da fábrica



FONTE: Colombo e Karnouskos (2011)

Outra tendência que deve ser considerada é a visão da “Internet das Coisas” (*IoT - Internet of Things*). Esta é uma tecnologia na qual os objetos formam uma rede para comunicação uns com os outros através da Internet, sem que haja interação humano-humano ou humano-computador. Este conceito vem sendo aplicado para indústrias sob denominação de IIoT (*Industrial Internet of Things*) e espera-se que ela transforme a realidade das indústrias de manufatura, óleo e gás, agricultura, mineração, transporte e saúde, que hoje correspondem a quase 2/3 da economia mundial (WORLD ECONOMIC FORUM, 2015). Consiste em sensores, atuadores, sistemas embarcados, máquinas e robôs interconectados uns com os outros, com os sistemas corporativos e também com a internet. Segundo levantamento feito pela (SAP, 2015) os especialistas preveem para o ano de 2020, mais de 30 bilhões de dispositivos, aparelhos, máquinas e outros objetos físicos estarão conectados. Comunicando-se através da Internet, estima-se que esses objetos irão gerar 403 trilhões de gigabytes de dados por ano até 2022. Nesse

período, a IIoT deverá gerar US$14,4 trilhões em valor combinado de receita e redução de custos.

Apesar desta tendência de evolução na interligação e comunicação entre dispositivos, entretanto, levando em consideração a realidade da indústria brasileira, percebe-se uma grande discrepância entre a situação atual, em nível de qualidade e tecnologia, se comparada a perspectiva global dos países desenvolvidos. Um primeiro passo essencial antes de almejar o estado da arte na tecnologia de interconexão dos processos industriais com os processos gerenciais e com a internet, consiste em desenvolver ferramentas e meios para disponibilizar os dados das máquinas de maneira padronizada e completa a toda organização.

1.1 **Justificativa**

Apesar de carecer de estudos e levantamentos estatísticos, pode-se afirmar que a maior parte do maquinário do chão de fábrica no Brasil é composto por máquinas-ferramenta: centros de usinagem, CNC, injetoras, extrusoras, corte a LASER e similares. A comunicação serial (RS-232/422/485) com protocolo MODBUS têm sido tradicionalmente utilizada nestes equipamentos e numa vasta gama de dispositivos industriais, incluindo CLPs, IHMs, instrumentos, medidores, *drivers* de acionamento e na maioria das aplicações de automação industrial (TEXAS INSTRUMENTS, 2014). O grande volume de produtos com estas configurações demonstram que este padrão de comunicação serial é forte em muitas áreas da indústria devido ao processamento mínimo exigido, a robustez e confiabilidade de conectores. Mesmo novos produtos continuam a adotar RS-232 e RS-485.

Entretanto, continuar acessando estes produtos através destas interfaces seriais ponto a ponto, além de dificultar a implementação de sistemas de integração com os níveis corporativos, pode se tornar um desafio ao longo do tempo. Os computadores mais recentes, especialmente notebooks, não possuem mais portas seriais (RS-232/422/485), e uma conexão serial é limitada pelo comprimento do cabo (geralmente 10 m).O catálogo de máquinas CNC atuais como Siemens e ROMI e de injetoras da fabricante LS como mostra (MOREIRA, 2010), demonstram que as máquinas atuais já incluem porta de comunicação Ethernet nos seus produtos, indicando que esta é uma forte tendência para o futuro recente. Além de já ser o padrão de rede utilizado nos escritórios e por já possuir infra-estrutura instalada em quase todas as empresas, o baixo custo do hardware faz da Ethernet uma opção atraente para aplicações de redes industriais. Praticamente todos os computadores pessoais hoje possuem uma placa de rede Ethernet instalada.

Diante desse cenário, justifica-se a proposta deste trabalho em avançar na integração vertical dos processos das organizações, através da interligação das máquinas do chão de fábrica à rede corporativa. Foi proposto assim, realizar implementação de um sistema embarcado que adquira dados de máquinas e equipamentos industriais, disponibilizando estes dados na rede local corporativa.

1.2 **Objetivos**

1.1.1 Objetivo Geral

Pesquisar, projetar e implementar um sistema embarcado para aquisição de dados de máquinas e equipamentos industriais do chão de fábrica, disponibilizando os mesmos na rede corporativa das empresas.

1.1.2 Objetivo Específicos

- Disponibilizar dados e informações importantes do chão de fábrica aos gerentes da organização;
- Melhorar a eficiência operacional por meio do gerenciamento remoto das máquinas;
- Buscar maior interação e colaboração entre homens e máquinas, com a melhoria de produtividade e do ambiente de trabalho;

1.3 **Limitações do Trabalho**

Neste trabalho será utilizado kits de desenvolvimento para microcontroladores para realizar as implementações e simulações do sistema embarcado proposto, portanto não pretende-se projetar hardware especifico para a solução.

Apesar da questão de segurança de dados ser importante ao tratarmos de disponibilização de dados em redes de comunicação, este trabalho não irá abordar este problema.

2 INTEGRAÇÃO GERÊNCIA/CHÃO DE FÁBRICA

2.1 Realidade Atual das Organizações

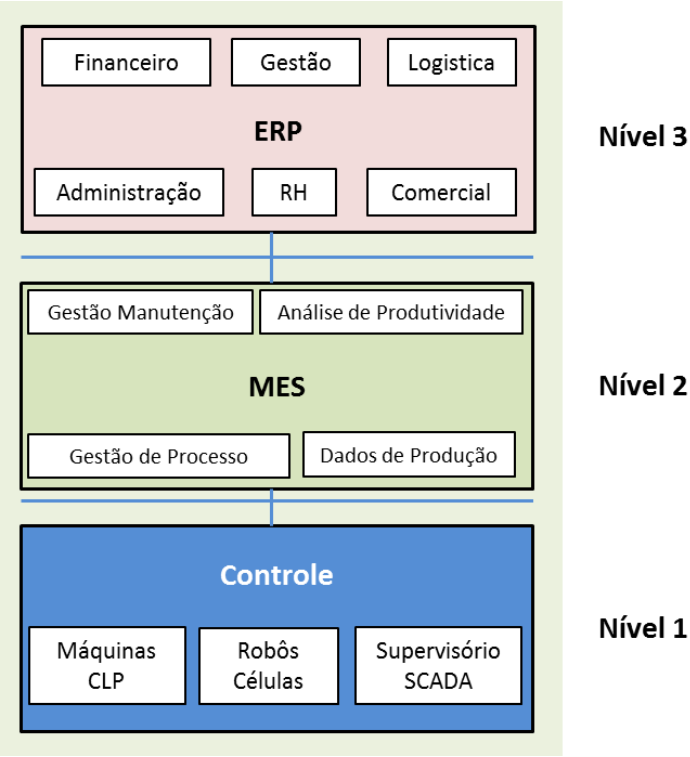
Desde o desenvolvimento da internet e sua consequente utilização na automação industrial, a integração de sistemas industriais tem se tornado um desejo crescente. Como aborda (SAUTER, 2007), apesar desta demanda ser recente, pode-se definir que o conceito de integração na automação teve sua formulação em 1970, com a criação do modelo CIM (*Computer-Integrated Manufacturing*). Esta ideia propunha integrar diversas ilhas de sistemas especializados a um sistema único e transparente de processamento de informações, padronizado e distribuído para toda a organização. Entretanto, considera-se que as altas expectativas em relação a este modelo nunca foram razoavelmente atendidas.

Durante a última década, os sistemas ERP trouxeram uma mudança de paradigma sob o ponto de vista de integração da informação de negócio das organizações. Diferente dos primeiros esforços na década de 70, os sistemas ERP conseguiram explorar o grande avanço da Tecnologia de Informação (TI) para atingir a integração total dos processos das organizações, incluindo fornecedores e clientes. Por outro lado, não existia uma padronização de funções e recursos, dependendo o sucesso de implementação a cada sistema particular. Assim, o ERP não era capaz de acessar as informações ao nível de chão de fábrica. Para esta finalidade surgiram os sistemas MES, responsáveis por criar esta ponte entre o nível corporativo e os níveis inferiores.

Em resumo, a composição típica teórica das organizações do ponto de vista da informação é uma hierarquia de três níveis (ERP, MES, Controle) como mostra a Figura 2-1. Apesar desta definição, no mundo real a situação é diferente. A integração entre ERP e MES atualmente se mostra confiável, com grandes esforços em padronizar os dados através do ISA-95 (adotada em nível internacional como IEC 62264) definindo a interface entre as funções de controle e outras funções empresariais (IEEE INDUSTRIAL ELECTRONICS MAGAZINE, 2007). Já a comunicação entre o sistema MES e o nível de controle ainda apresenta muitos problemas, principalmente devido ao fato de que estes dois níveis trabalham com

padrões diferentes. Enquanto no escritório predomina as redes LAN, no nível de controle predomina os barramentos de campo ou *Fieldbus*.

Figura 2-1 - Níveis hierárquicos teóricos da organização, sob o ponto de vista da tecnologia da informação



FONTE: O AUTOR (2015)

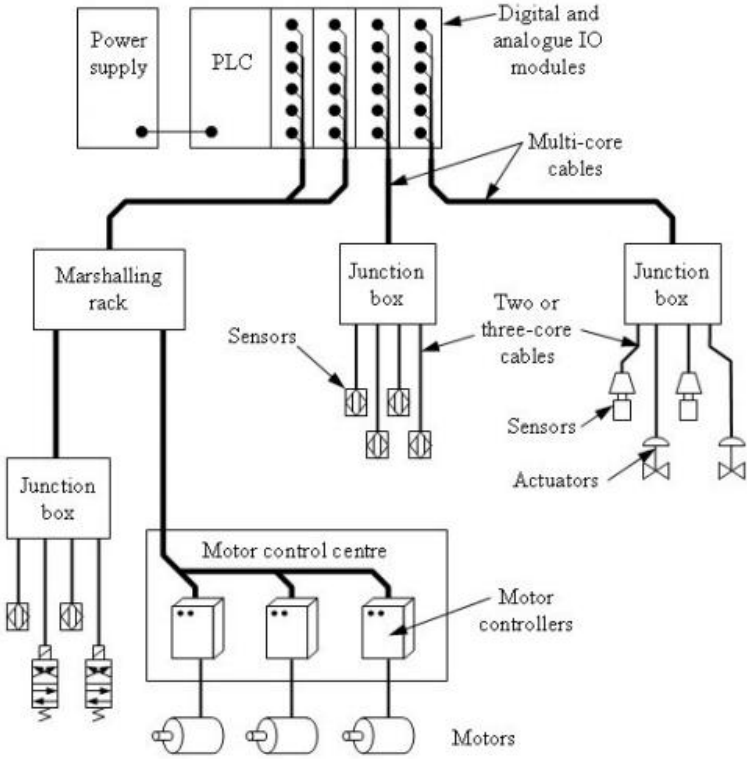
2.2 Comunicação de Dados na Indústria

2.2.1 Nível 1 – Nível de Controle

O nível de controle da organização é o ponto mais crítico para a integração. Este nível está relacionado ao controle do chão de fábrica, composto por robôs e máquinas que possuem sensores e atuadores conectados às entradas e saídas de um controlador e são normalmente instalados na planta. Assim, as conexões de um sistema de controle tradicional são realizadas ponto a ponto e exigem cabos com longas distâncias. Pode-se observar na arquitetura típica de comunicação na

indústria (Figura 2-2) que esse tipo de instalação é complexo em termos de manutenção e de verificação de falhas.

Figura 2-2 - Conexão ponto a ponto tradicional

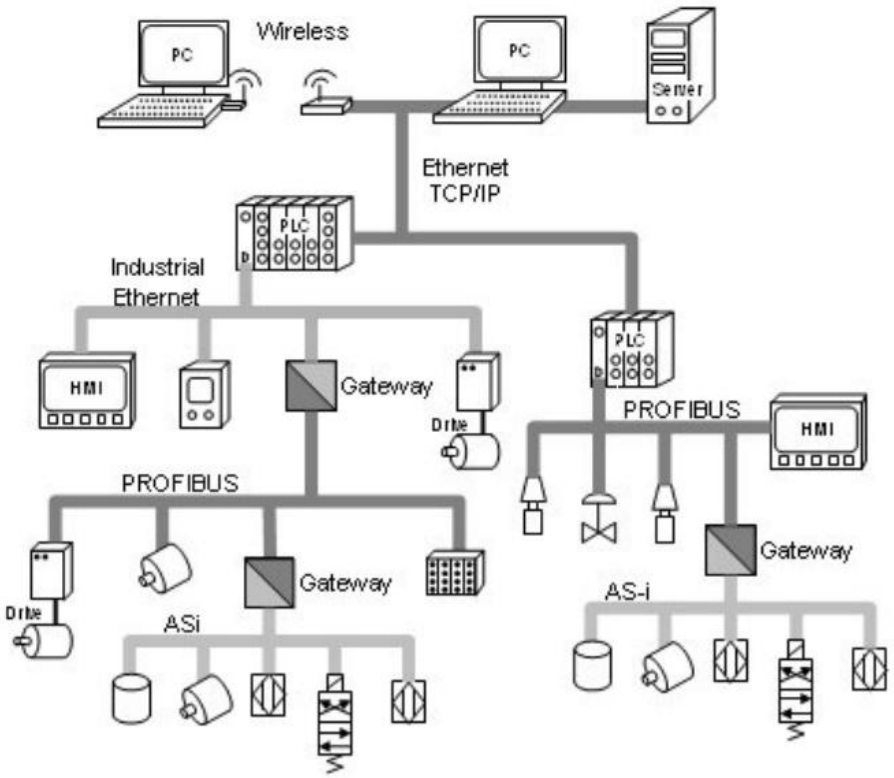


FONTE: VERWER (2014)

Para substituir a ligação ponto-a-ponto entre cada sensor e atuador e o controlador, foi desenvolvido o barramento de campo, ou *Fieldbus,* para estabelecer um padrão de comunicação entre controladores industriais e os sensores e atuadores instalados em campo (ALBUQUERQUE,2009). Assim, comparado ao sistema tradicional, são adicionados módulos de entradas e saídas próximos aos sensores e atuadores, sendo os dados transmitidos ao controlador através de um cabo F*ieldbus* conforme mostra a Figura 2-3.

Há muitos tipos diferentes de barramentos de campo disponíveis. A maioria definidos por normas internacionais e suportados por diversos fabricantes. Pode-se citar os mais utilizados na industria como exemplo: Profibus, DeviceNet, CAN, Modbus e AS-i. Estes são utilizados dependendo da área de aplicação, dos requisitos necessários e dos gastos envolvidos.

Figura 2-3 - Visão geral do nível de controle e suas interligações



FONTE: VERWER (2014)

Os tipos de *Fieldbus* geralmente não possuem compatibilidade uns com os outros. Isso significa que todos os dispositivos devem utilizar o mesmo padrão de comunicação de dados. Caso seja necessário integrar tipos diferentes de barramentos de campo, *gateways* fornecem a conversão de uma tecnologia para outra como mostra a Figura 2-3. Na prática, são utilizados vários padrões de barramentos de campo. Eles dependem uns dos outros, cada um executando funções que se complementam e até, muitas vezes, se sobrepõem. Em relação a este fato específico, não existe uma definição clara, tornando as interligações extremamente complexas e dificultando a sua implementação de uma forma simples e padronizada. Com a maioria dos fabricantes de máquina adotando uma visão pragmática, as comunicações de dados no nível de controle foram implementadas como “ilhas” especializadas que não conversam umas com as outras, horizontalmente.

2.2.1.1 Modbus

Dentre os diversos protocolos *Fieldbus* existentes, pode-se destacar o protocolo Modbus. Este é um protocolo aberto, padronizado, com comunicação em tempo real de grande quantidade de dados, de baixo custo e tem sido amplamente utilizado em CLP's, controladores, *drivers* de acionamentos e instrumentos. Ele suporta meio físico através de padrão RS-232, RS-422, RS-485 e Ethernet. A troca de mensagens é do tipo Mestre-Escravo, ou seja, o dispositivo mestre envia uma solicitação para o escravo que responde (MODBUS, 2012).

Qualquer esforço para coletar os dados de máquinas do chão de fábrica deve prever o suporte a protocolos de comunicação, visando facilitar a troca de dados com estes equipamentos. Pelo grande aumento na utilização do protocolo Modbus nos últimos anos, especialmente após a implementação do Modbus TCP (VILELA, 2013), e pelas características citadas, considera-se Modbus como o protocolo ideal para realizar a abordagem de interligação com máquinas e equipamentos industriais. Além disso, como a distância de interligação entre equipamentos é um requisito importante, torna-se mais vantajoso a utilização do padrão RS-485 que permite uma distância entre os pontos de conexão de até 100m. O padrão serial RS-485 continua sendo o meio físico mais utilizado atualmente para este protocolo.

2.2.1.2 Máquinas do Chão de Fábrica

O chão de fábrica da indústria brasileira é caracterizado por uma natureza diversa de máquinas e equipamentos. Pode-se considerar que uma parcela destas máquinas são modelos padronizados de produção seriada, como por exemplo: centros de usinagem, extrusoras e injetoras. Estas podem possuir protocolos de comunicação ou não, dependendo do fabricante e modelo, uma vez que a possibilidade de fornecer integração através de protocolos de comunicações é um diferencial para estes fabricantes. Na maior parte dos casos, os protocolos de comunicação implementados neste tipo de equipamento são proprietários.

Outra parte das máquinas são desenvolvidas por fabricantes de máquinas sob demanda, com projeto único e específico para atender uma função determinada e são focadas na execução e automação de operações de manufatura. Mesmo possuindo controladores e sistemas automatizados de alto nível tecnológico, não

existe necessariamente a preocupação específica com a transferência dos dados. Por fim, existem máquinas que são extremamente simples, com operações manuais, não possuem controladores ou tecnologia embarcada de automação, sem opção de coleta de dados através de protocolos de comunicação, exceto diretamente através de sinais digitais e analógicos.

De uma maneira geral, a maioria das máquinas que possuem controladores industriais disponibiliza alguma porta para comunicação. Na prática, porém, não são utilizados para interligação se não houver uma demanda específica do cliente. Ocorre geralmente que, no futuro, quando o cliente tem interesse em realizar integração com o nível corporativo, ele solicita ao fabricante da máquina uma adaptação para adicionar este recurso.

Além disso, os interesses mais comuns envolvem a implementação de sistemas supervisórios (SCADA), que ficam dedicados em um único computador de supervisão e acessam ponto a ponto apenas uma máquina. Este tipo de implementação é caracterizada como pertencente ao nível 2.

2.2.2 Nível 2 – Nível de Monitoramento e Supervisão

O nível 2 corresponde ao nível de monitoramento e supervisão das plantas ou células de trabalho. Tradicionalmente, os próprios padrões *Fieldbus* tem sido utilizados para este fim. Porém, a interligação vertical geralmente limita-se ao ambiente de Supervisão e Aquisição de Dados (SCADA).

Alternativas normalmente utilizadas para implementar a integração, consiste na utilização de um computador centralizado para adquirir os dados de diversas máquinas. Entretanto, o computador tem diversos recursos, capacidade de processamento e periféricos que são dispensáveis neste tipo de aplicação, tendo um custo elevado. Além disso, seriam necessárias interfaces para aquisição de dados digitais e muitas portas seriais para adquirir através de comunicação, tornando essa solução inviável.

Outra alternativa, a utilização de gateways para conversão de protocolo são encontrados no mercado. Entretanto, seria necessário esforços para desenvolvimento de softwares com o objetivo de gerenciar a coleta dos dados. Outro

fator que deve ser levado em consideração são as máquinas que não oferecem possibilidade de interface através de protocolo de comunicação. Neste caso, outra abordagem deveria ser realizada. Uma interligação do nível 2 com as redes de nível superior, tais como LANs, raramente são previstas e dependem exclusivamente de soluções com *gateway*s dedicados.

.

2.2.3 Nível 3 – Nível Corporativo

O nível 3 compreende os dispositivos para os escritórios e instalações destinadas ao gerenciamento da planta e das informações organizacionais. Neste âmbito, predomina o padrão Ethernet e também as redes sem fio. Quando deseja-se obter dados do chão de fábrica, opta-se por fazer coletas manuais, através do preenchimento de relatórios e planilhas. Quando algum grau de automação é desejável para o processo, são adquiridos sistemas que embarcam o seu método de coleta e interligação, com protocolos de comunicação proprietários. Assim cria-se uma infra-estrutura de rede separada apenas para a coleta de dados no nível de chão de fábrica.

Neste contexto, pode ser verificado que a implantação de soluções específicas de integração, precisariam de esforços isolados para cada tarefa, levando a uma infra-estrutura que não é interoperável devido as barreiras causadas por padrões diferentes de dados. O resultado é a existência de tipos de tecnologia que complicam ainda mais a integração e o nível de complexidade de todo o sistema. As abordagens atuais para comunicação de dados dentro da na indústria apontam o tendências diferentes para interligação da organização como um todo.

2.3 Abordagens Modernas para Comunicação de Dados na Industria

Do ponto de vista das informações, há uma tendência para a redução da hierarquia vista na Figura 2-1, onde são observadas estruturas restritas em níveis, com uma camada fornecendo informações para a camada superior, até atingir o topo da hierarquia. Nos últimos anos, os esforços buscam a centralização da informação com conceitos modulares e flexíveis tornando-se cada vez mais populares.

Esta tendência é reforçada, pelas tecnologias que vem sendo utilizadas na área de TI, especificamente na infra-estrutura de redes, onde pode-se perceber uma centralização de recursos. Serviços de TI são acessados através da Internet, assim como ferramentas e aplicações oferecem a sensação de estarem instalados localmente (geralmente através de um navegador web). Considerando que as indústrias estão buscando adotar estratégias de integração de dados e, como essa integração depende integralmente dos recursos de TI instalados nestas indústrias, naturalmente estas tendências estão influenciando a forma como as aplicações industriais são projetadas.

Uma visão de como os sistemas industriais da próxima geração podem parecer está representado na Figura 2-4. Identifica-se uma infra-estrutura onde a informação flui de maneira plana (sem hierarquias), todos os sistemas como ERP, CLP's, SCADA, MES e as máquinas, estão conectados a toda informação centralizada disponível na organização. Assim, cada integrante dessa rede será capaz de cooperar, compartilhar informações e ser um elemento ativo de um sistema mais complexo. Apesar da mudança de paradigma proposta, esse modelo não exclui ainda a dificuldade existente em coletar os dados das máquinas do chão de fábrica, e armazená-las nos servidores de informações centralizados, estejam eles instalados localmente ou acessíveis na internet.

2.4 Proposta Moderna para Aquisição e Criação de Servidor de Dados

Baseado nas abordagens modernas para integração de dados, o objetivo deste trabalho consiste em buscar uma solução ao problema de acesso aos dados do chão de fábrica por parte dos níveis corporativos. Ao mesmo tempo, busca-se implantar novos paradigmas de interconexão e centralização de dados. Portanto, a proposta do trabalho é desenvolver um sistema embarcado que seja capaz de coletar dados de máquina ponto-a-ponto e disponibilizá-los através de um servidor que roda embarcado no próprio sistema;

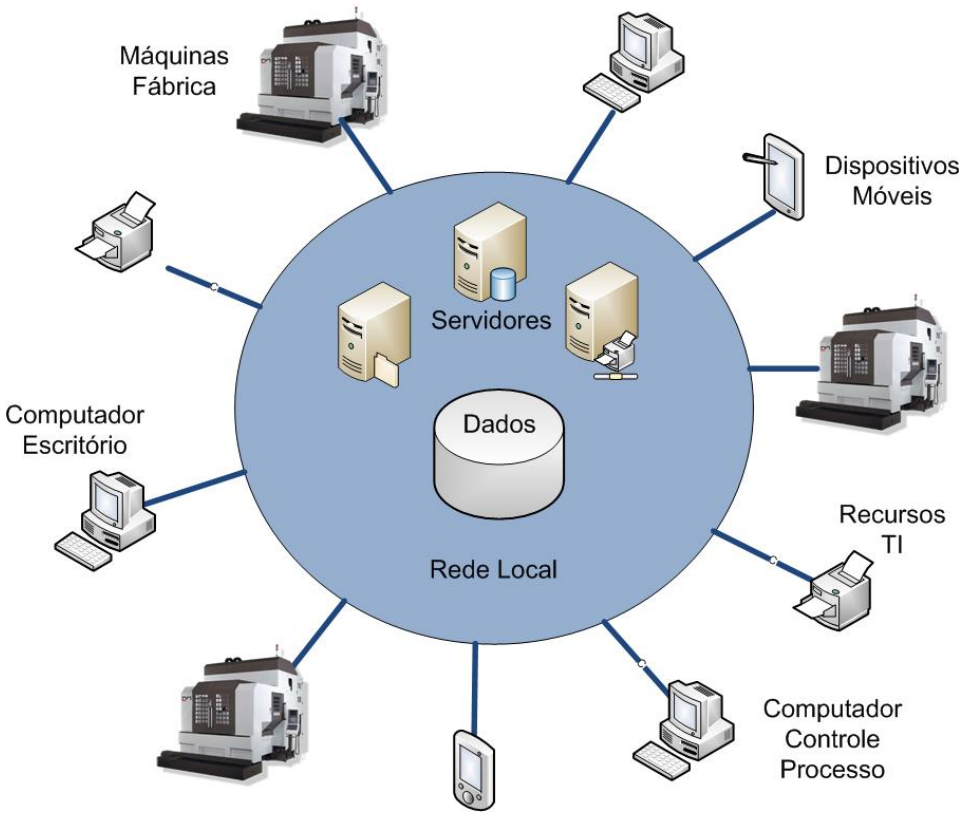
Os dados que pretende-se coletar correspondem a:

- **Sinais digitais** provenientes de sensores ou botões, que possam indicar máquina em funcionamento/em parada ou para contagem de peças produzidas;

- **Sinais analógicos** para coleta de grandezas físicas através de sensores e transdutores;

- Informações através de rede de comunicação **Modbus Serial RS-485,** para adquirir dados de máquinas que disponibilizem este recurso.

Figura 2-4– Tendência Arquitetura da Organização



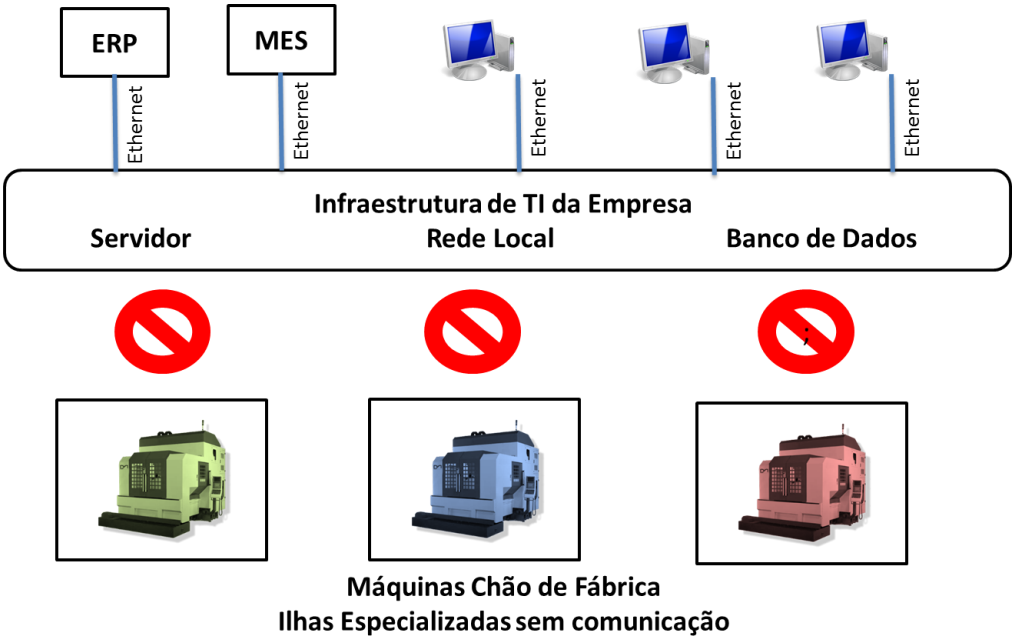
FONTE: O AUTOR (2015)

Após a coleta, o sistema deve disponibilizar estes dados através de um servidor embarcado no próprio sistema, podendo ser acessado na rede corporativa local através do padrão Ethernet TCP/IP. Este servidor ficará disponibilizando as informações para todos os integrantes da rede de uma forma plana, sem depender de hierarquias.

A Figura 2-5 mostra como seria a implementação hoje, de uma arquitetura onde a informação está centralizada dentro da empresa. Neste esquema fica evidente que as máquinas atuais não possuem conexão com a rede corporativa, e

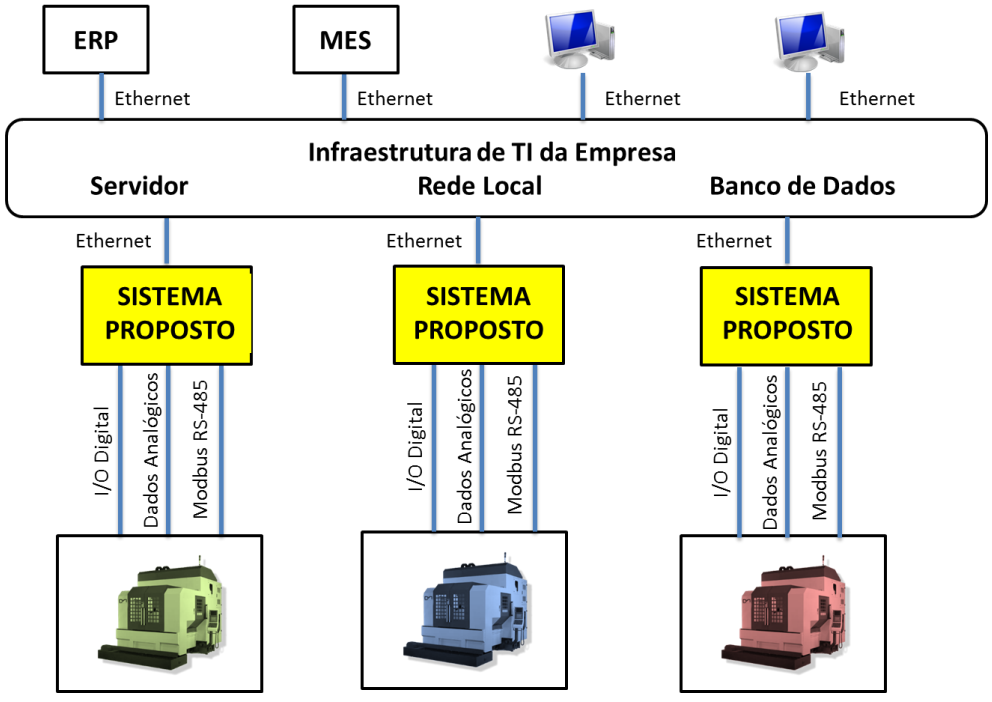
trabalham como ilhas especializadas. Enquanto isso, a Figura 2-6 demonstra a proposta do trabalho para interligação das máquinas a rede local, indicando onde o sistema proposto será inserido nesta estrutura.

Figura 2-5 – Arquitetura proposta dentro da situação atual, destacando o problema de interligação das máquinas



FONTE: O AUTOR (2015)

Figura 2-6 - Demonstração da arquitetura de rede com inclusão do sistema proposto



FONTE: O AUTOR (2015)

Analisando a proposta apresentada, pode-se perceber que a informação do chão de fábrica não necessariamente precisa passar por um nível intermediário (MES) para que as informações atinjam os computadores dos gerentes da organização. Todos os dados das máquinas ficam centralizados, conectados a infra-estrutura de rede de TI. Claramente, esta proposta tem diversas vantagens como:

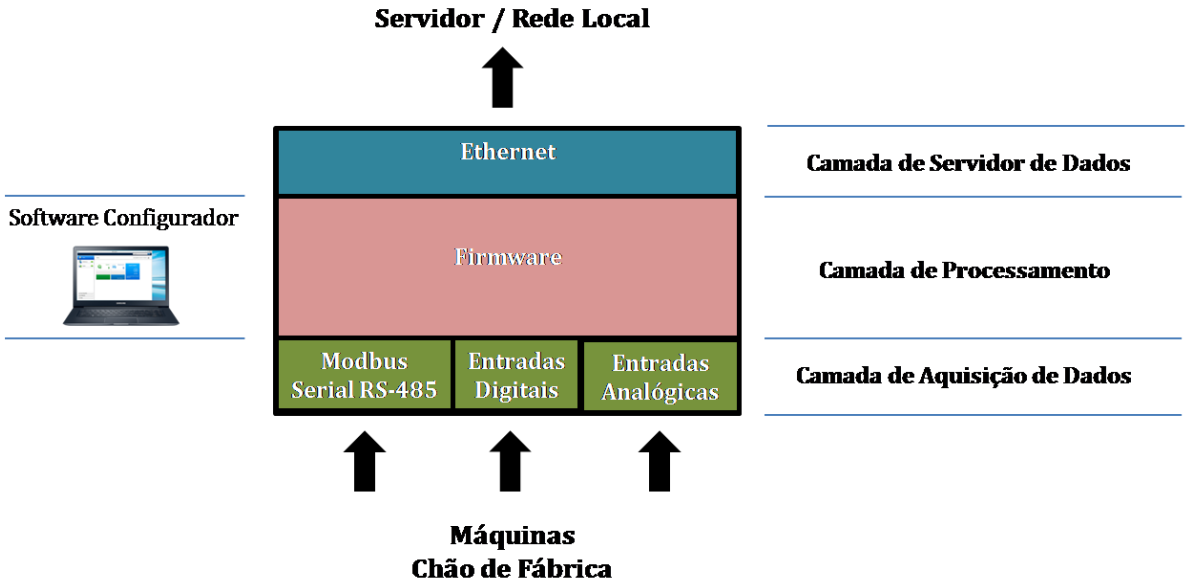
- Informação centralizada, os dados ficam disponíveis para toda organização através da rede corporativa, sem depender de adaptações ou níveis intermediários;

- Aproveitamento da infra-estrutura de rede de TI existente nas indústrias, ao invés de soluções com gateways ou protocolos proprietários que exige instalação de outras estruturas de rede, com custo maior.

- Os dados que estão na rede local podem ser transferidos facilmente para internet, ficando disponíveis para serem acessados no mundo todo.

A Figura 2-7 mostra a arquitetura proposta para o sistema embarcado proposto denominado “Interface Integradora”. Pode-se perceber que este sistema será constituído por três camadas: Camada de aquisição de dados, camada de processamento e camada de interligação com o servidor da organização.

Figura 2-7 - Arquitetura proposta do sistema embarcado para coleta de dados



FONTE: O AUTOR (2015)

2.4.1 Camada de Aquisição de Dados

A primeira camada denominada “Camada de Aquisição de Dados”, será responsável por adquirir os dados das máquinas do chão de fábrica. Como mencionado no item 2.4, os dados serão adquiridos através de três interfaces distintas: sinais digitais, sinais analógicos e comunicação Modbus Serial RS-485.

Esta camada demanda como requisitos, que um microcontrolador possua recursos embutidos de comunicação serial MODBUS através de interface UART RS485 para realizar interligação com as máquinas,. Além disso, também deverá conter pinos de sinais digitais e pinos de entrada para sinais analógicos com conversor analógico/digital.

2.4.2 Camada de Processamento

O sistema embarcado deve incluir uma camada de firmware que será responsável por gerenciar a coleta de dados das máquinas e o envio destes para a rede local. Este recurso diferencia o sistema proposto de um simples *gateway* que converte dados de um protocolo para o outro. Ao incluir esta camada de software, tem-se a vantagem de tornar ter toda a solução embarcada em um único dispositivo.

Inevitavelmente, cada máquina onde o sistema será instalado necessitará de parametrização e configuração para funcionar de acordo com cada caso. Para isso, será proposto também o desenvolvimento de um software baseado em PC para configuração da Interface Integradora. Com o software configurador instalado em um computador que esteja conectado na rede local ou ponto-a-ponto, será disponibilizada a parametrização do sistema. Estes os parâmetros disponíveis serão: origem dos dados (sinal digital, analógico, ou MODBUS RS-485), endereços dos registradores Modbus para leitura de baixo nível e canais de aquisição digitais e analógicos. Assim, tem-se um sistema genérico e flexível que pode ser facilmente adaptado e configurado a qualquer organização e para diferentes máquinas, transformando em um produto facilmente comercializável em larga escala.

2.4.3 Camada de Servidor de Dados

Esta camada será responsável por preparar os dados que foram adquiridos na camada de aquisição de dados e, de acordo com a configuração realizada na camada de processamento, disponibilizar as informações através de um servidor. Esta camada exige que o microcontrolador possua conexão Ethernet para interligação com o servidor de dados da rede local.

3 DESENVOLVIMENTO INTERFACE INTEGRADORA

O primeiro passo para iniciar a implementação da Interface Integradora é a escolha do hardware. Deve ser utilizado um microcontrolador que possua os recursos exigidos para a aplicação de acordo com a descrição realizada em relação aos requisitos de cada camada do sistema.

3.1 Definição do Hardware

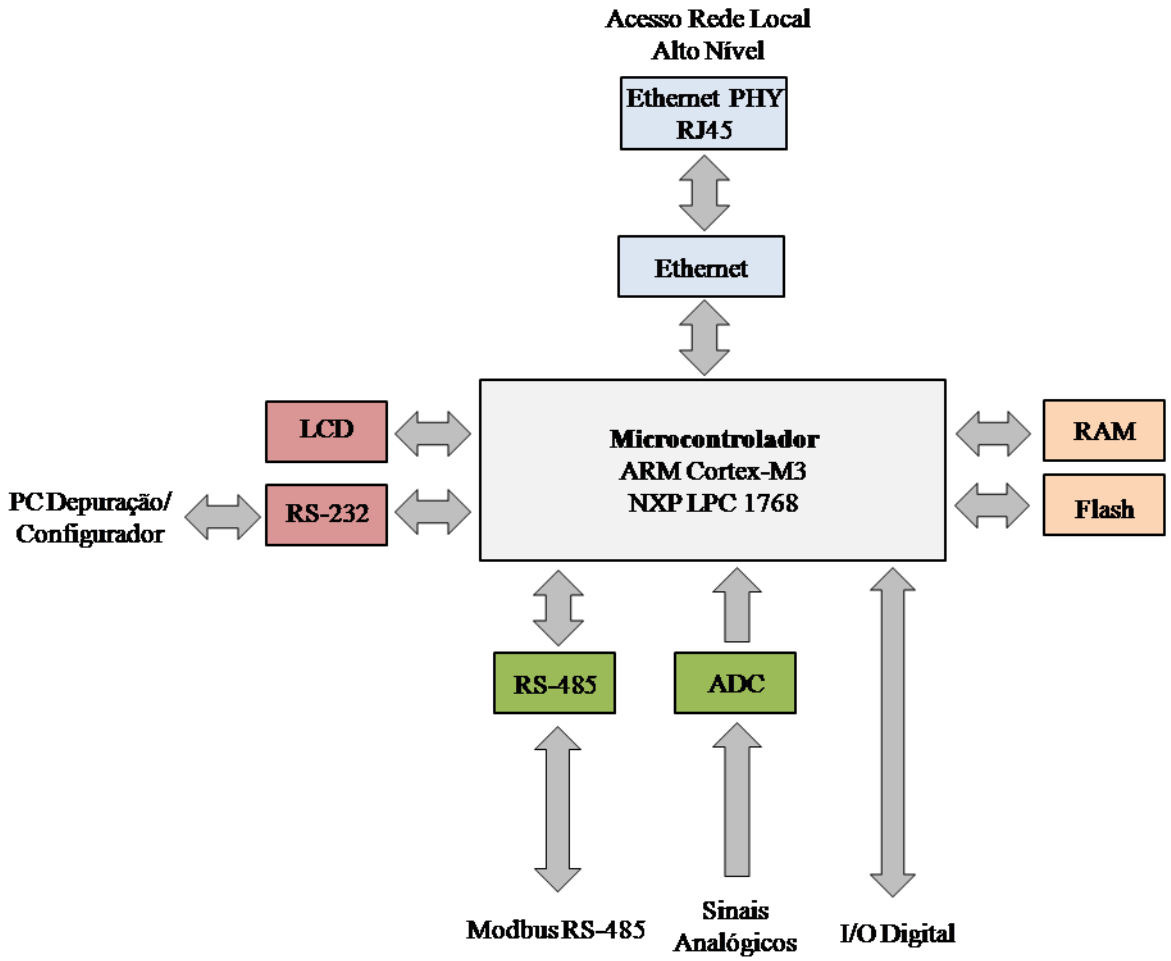
Com os requisitos necessários verificados previamente, foi escolhido para a implementação do sistema proposto um microcontrolador ARM Cortex-M3 do fabricante NXP® modelo LPC1768. A escolha foi reforçada pelo fato ter a disposição uma placa de desenvolvimento disponível para este microcontrolador. Assim, todo o projeto será desenvolvido utilizando o kit de desenvolvimento MCB1700 desenvolvido pela Keil®.

Em relação aos requisitos da aplicação, para a camada de aquisição de dados o LPC1768 possui 70 pinos de I/O (GPIO) para aquisição de sinais digitais, além de 8 pinos com conversão analógico-digital (ADC) de 12 bits de resolução. Para a aquisição através de comunicação, o microcontrolador conta com 3 portas de comunicação serial UART (*Universal Asynchronous Receiver/Transmitter*), sendo uma com suporte ao padrão RS-485 que será utilizado para adquirir sinais via protocolo de comunicação MODBUS RS-485.

Para a camada de processamento, o microcontrolador possui uma CPU que roda com uma frequência de 100MHz e possui 64 kBytes de memória RAM e 512 kbytes de memória flash não-volatil. Além disso, possui sistema de gravação do programa através de uma porta serial RS-232 ISP (*In-System Programming*) sem necessidade de adquirir gravadores específicos.

Para a camada de Servidor de dados, o LPC 1768 possui embutido um bloco completo com suporte a Ethernet 10 Mbits/s e 100 Mbits/s projetados para fornecer desempenho otimizado com o uso de aceleração de hardware DMA. A placa de desenvolvimento inclui um conector RJ-45 com adaptador PHY (camada física). Deste modo, o hardware está apto para implementação de comunicação Ethernet TCP/IP.

Figura 3-1 - Esquema do hardware do microcontrolador LPC 1768



FONTE: O AUTOR (2015)

3.1.1 Ferramentas de Software de Programação

A utilização do ARM da NXP® LPC 1768 implica a utilização de um software para a programação do microcontrolador. Existem diversas opções disponíveis de ambientes de desenvolvimento integrado (IDEs) tanto dá NXP como de terceiros. Como o kit de desenvolvimento utilizado é da fabricante Keil® e a mesma possui uma IDE chamada uVision®, dedicada ao desenvolvimento para microcontroladores ARM, justificando-se a escolha deste para a implementação do trabalho. A linguagem de programação utilizada na solução foi a linguagem C.

Portanto, o microcontrolador escolhido fornece todos os recursos necessários para a aplicação e o desenvolvimento da Interface Integradora foi iniciado pela camada de aquisição de dados.

3.2 Implementação da Camada de Aquisição de Dados

Na camada de aquisição de dados, os dados da máquinas devem ser adquiridos através da leitura dos sinais digitais pela porta GPIO do microcontrolador. Para realizar a aquisição dos sinais analógicos, deve-se utilizar os canais conectados ao conversor analógico digital. Tanto para sinais digitais quanto analógicos, não é necessário o desenvolvimento de nenhuma função especializada de baixo nível, pois o uVision já possui funções internas no ARM para estas finalidades.

Por outro lado, a aquisição dos sinais através de MODBUS RTU Serial RS-485 deve ser implementado no microcontrolador de acordo com as especificações do protocolo.

3.2.1 Modbus RTU Serial RS-485

O Modbus é um protocolo de camada de aplicação que fornece comunicação cliente/servidor entre dispositivos conectados em tipos de barramentos ou redes. Em um meio físico serial, o papel do cliente é realizado pelo mestre do barramento serial e os escravos agem como servidores. Esta padronização é implementada na Camada de Enlace (MODBUS, 2002).

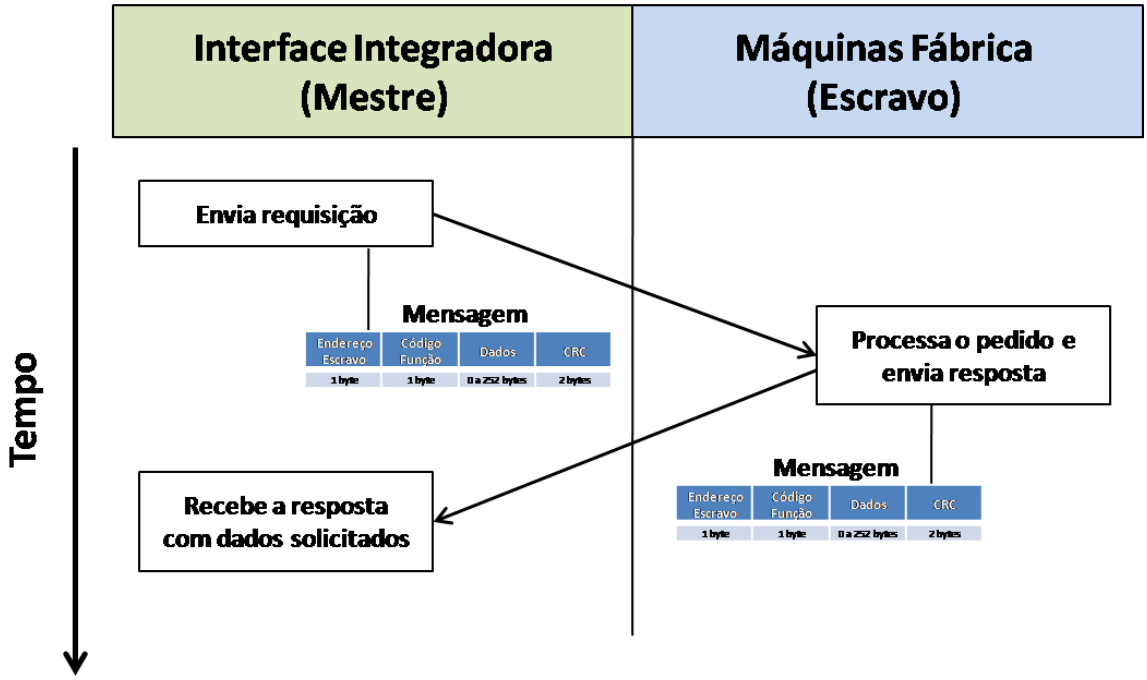
A Camada de Enlace de dados Modbus é composto por duas subcamadas separadas:

• O protocolo mestre / escravo

• O modo de transmissão (RTU *[Remote Terminal Unit]* ou ASCII)

Na Figura 3-2 pode-se analisar como é realizada a comunicação no protocolo Mestre/Escravo. O dispositivo mestre emite comandos para um escravo da rede, que por sua vez responde ao mestre conforme solicitado. Os escravos não transmitem dados sem um pedido do mestre, e não se comunicam com outros escravos. Na solução proposta, como a Interface Integradora necessita solicitar dados a outros dispositivos, a interface deve operar como um mestre da rede enquanto as máquinas do chão de fábrica devem ser escravos.

Figura 3-2 - Esquema de comunicação utilizado no protocolo colo Mestre/Escravo



FONTE: O AUTOR (2015)

O modo de transmissão, por sua vez, define a codificação da informação transmitida. O modo de transmissão padrão do protocolo conforme (MODBUS,2002) é o RTU. Quando os dispositivos comunicam em uma linha serial MODBUS usando o modo de transmissão RTU, cada byte em uma mensagem contém dois caracteres hexadecimais de 4 bits. O mesmo modo de transmissão RTU deve ser usado em todos os dispositivos em uma rede MODBUS. O formato da mensagem enviada no protocolo MODBUS RTU, também chamado de *frame,* é visto na Figura 3-3.

Figura 3-3 - Frame MODBUS RTU

Endereço Escravo	Código Função	Dados	CRC
1 byte	1 byte	0 a 252 bytes	2 bytes

FONTE: O AUTOR (2015)

- Endereço Escravo: O escravo possui um endereço que para ser válido deve estar entre 0 - 247 decimal. Um mestre coloca neste campo da mensagem o endereço do escravo que ele deseja se comunicar. Quando o escravo retorna a sua resposta, ele coloca seu próprio endereço neste campo, para informar ao mestre qual escravo está respondendo.

- Código Função: O código de função indica ao escravo que tipo de ação o mestre está solicitando. Dependendo da função utilizada, caso mais informações forem necessárias, estas são inseridas no campo de dados.O Modbus especifica diversos tipos de funções para leitura e escrita de bits ou bytes.

- Dados: Neste campo da mensagem, o mestre informa parâmetros adicionais que são necessários de acordo com a função utilizada. Para o escravo, este campo é utilizado para responder com os dados solicitados pelo mestre.

- CRC: O campo de verificação de erro é o resultado de um cálculo de verificação de redundância conhecido como CRC16 (*Cyclical Redundancy Checking* – 16 bits) que é realizado no conteúdo completo da mensagem.

Para a interface integradora, como o objetivo é a aquisição de valores numéricos e, como estes valores são armazenados nos equipamentos na forma de registradores, para realizar esta tarefa deve ser implementado no microcontrolador a função Modbus *Read Input Registers,* representada pelo número hexadecimal 04. Esse código de função é usado para ler o conteúdo de um ou vários registradores em um dispositivo escravo. Para esta função, a mensagem Modbus deve ser criada de acordo com o exemplo da Figura 3-4, respeitando a especificação do protocolo Modbus (MODBUS, 2002).

Na requisição do mestre, deve-se inserir:

- O endereço do escravo conectado a rede, o qual serão solicitados a leitura dos valores de registradores;

- O endereço do primeiro registrador do escravo, do qual o mestre deseja ser informado (separado em dois bytes, mais significativo primeiro). Obrigatoriamente, são lidos registradores em sequência a partir de um endereço inicial. Caso for desejado ler vários registradores que não estejam em sequência, deve ser lido um registrador de cada vez;

- A quantidade de registradores que serão lidos a partir do primeiro registrador;

- O cálculo CRC da mensagem.

Figura 3-4 - Exemplo requisição/resposta da função Read Input Registers

Mensagem	Requisição (Mestre)		Resposta (Escravo)	
	Significado	**Valor (Hex)**	**Significado**	**Valor (Hex)**
Endereço Escravo	Endereço da Máquina	01	Endereço Máquina	01
Endereço Função	Read Input Registers	04	Read Input Registers	04
Dados	Endereço Inicio MSB	01	Número Bytes dos Dados	04
	Endereço Inicio LSB	3D	Valor Registrador MSB (316)	10
	Qtd. Registradores MSB	00	Valor Registrador LSB (316)	A3
	Qtd. Registradores LSB	02	Valor Registrador MSB (317)	2F
			Valor Registrador LSB (317)	03

Requisição
Endereço Escravo = 1
Read Input Register
Endereço Inicio Registrador = 316
Quantidade Registradores = 2

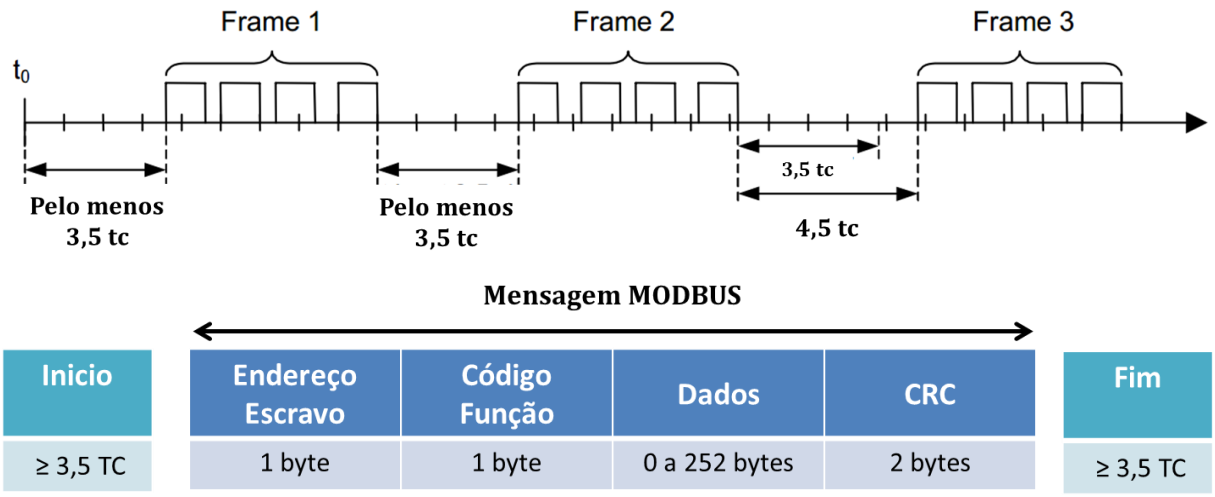
Resposta
Endereço Escravo = 1
Read Input Register
Número Bytes = 4
Valor Registrador (316) = 0x10A3 = 4259
Valor Registrador (317) = 0x2F03 = 12035

*MSB = Byte mais significativo
*LSB = Byte menos significativo

Os registradores possuem endereços que iniciam em zero. Portanto, quando são requisitados registradores endereçados de 1-16, na verdade o escravo interpreta como endereços 0- 15.

Um dispositivo que deseja transmitir, coloca uma mensagem Modbus em um quadro contendo um identificador padronizado para indicar o início e o final da mensagem. No modo RTU, as mensagens são separadas por um intervalo de silêncio de pelo menos 3,5 vezes o tempo de um caractere. Nas seções seguintes, esse intervalo de tempo é chamado 3,5 tc.

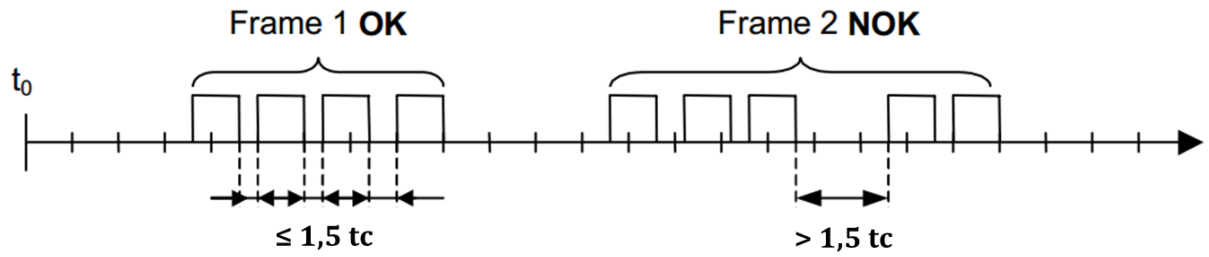
Figura 3-5 - Mensagem MODBUS RTU



FONTE: O AUTOR (2015)

Toda a mensagem deve ser transmitida como um fluxo contínuo de caracteres. Se um intervalo de silêncio com mais de 1,5 vezes o tempo de um caractere ocorrer entre dois caracteres, a mensagem é considerada incompleta e deve ser descartada pelo receptor.

Figura 3-6 - Tempo entre caracteres na comunicação Modbus

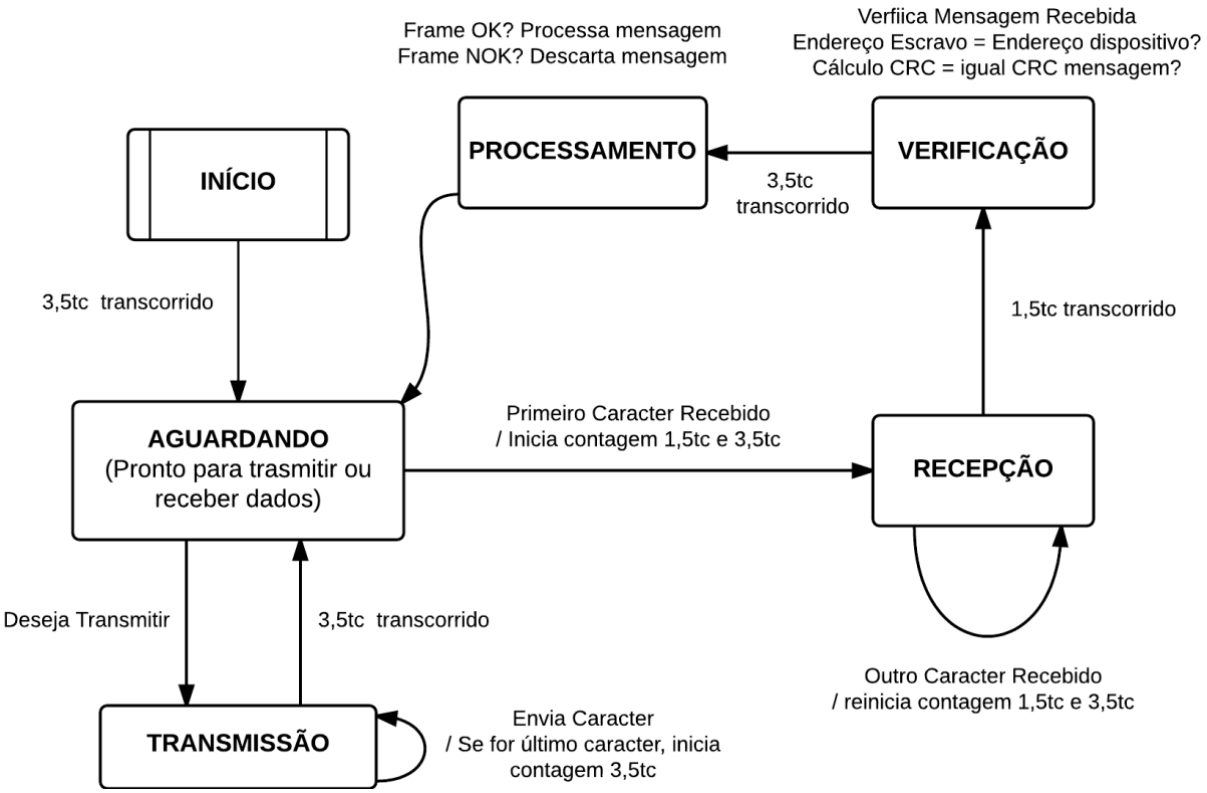


FONTE: O AUTOR (2015)

Para a implementação do Modbus RTU na interface integradora, o mestre deve gerenciar interrupções para os tempos de caracteres 1,5tc e 3,5tc. De acordo com o padrão determinado pelo protocolo, será utilizado uma taxa de transferência de 19200*bauds*, assim, devem ser utilizados valores fixos para os 2 temporizadores: recomenda-se o valor de 750µs para o tempo entre caracteres (1,5tc) e um valor de 1,75ms para a o tempo entre mensagens (3,5tc) (MODBUS, 2002).

O esquema da Figura 3-7 mostra a lógica de funcionamento da camada de aplicação do protocolo MODBUS RTU, tanto para o mestre como para o escravo, representada por um diagrama de estados.

Figura 3-7 - Diagrama de Estados do Modbus RTU



FONTE: O AUTOR (2015)

Ao energizar o dispositivo, o estado Modbus vai para "INICIO". A troca para o estado "AGUARDANDO" ocorre após um estouro do temporizador 3,5tc. O estado "AGUARDANDO" é o estado normal quando nenhuma transmissão ou recepção está ativa. Neste estado, qualquer caractere detectado na comunicação é identificado como o início da mensagem, levando para o estado “RECEPÇÃO”. Em seguida, os caracteres são recebidos até identificar um intervalo de tempo de 1,5tc sem nenhum caractere sendo recebido. Após, é realizado o a verificação do CRC e do endereço do escravo que define se a mensagem está correta e é destinada ao dispositivo. Em caso afirmativo, é realizado o processamento desejado, caso contrário a mensagem é descartada.

3.2.1.1 Implementação do Software da Camada de Aquisição

A implementação do protocolo Modbus abrange o desenvolvimento de uma máquina de estados como demonstrado no capítulo anterior, responsável por gerenciar a troca de mensagens. A Interface Integradora deve ser o mestre da rede, conectada a uma máquina escravo, adquirindo os dados através do envio mensagens utilizando a função Modbus *Read Input Register.* Como visto na definição do protocolo, esta função deve ser capaz de enviar uma solicitação de leitura de um registrador para o dispositivo escravo, receber a resposta e fazer a verificação de integridade dos dados.

Com o objetivo de obter uma melhor organização do código, foi criada uma estrutura global chamada **MbRTU** conforme padrão da mensagem Modbus mostrada na Figura 3-3.

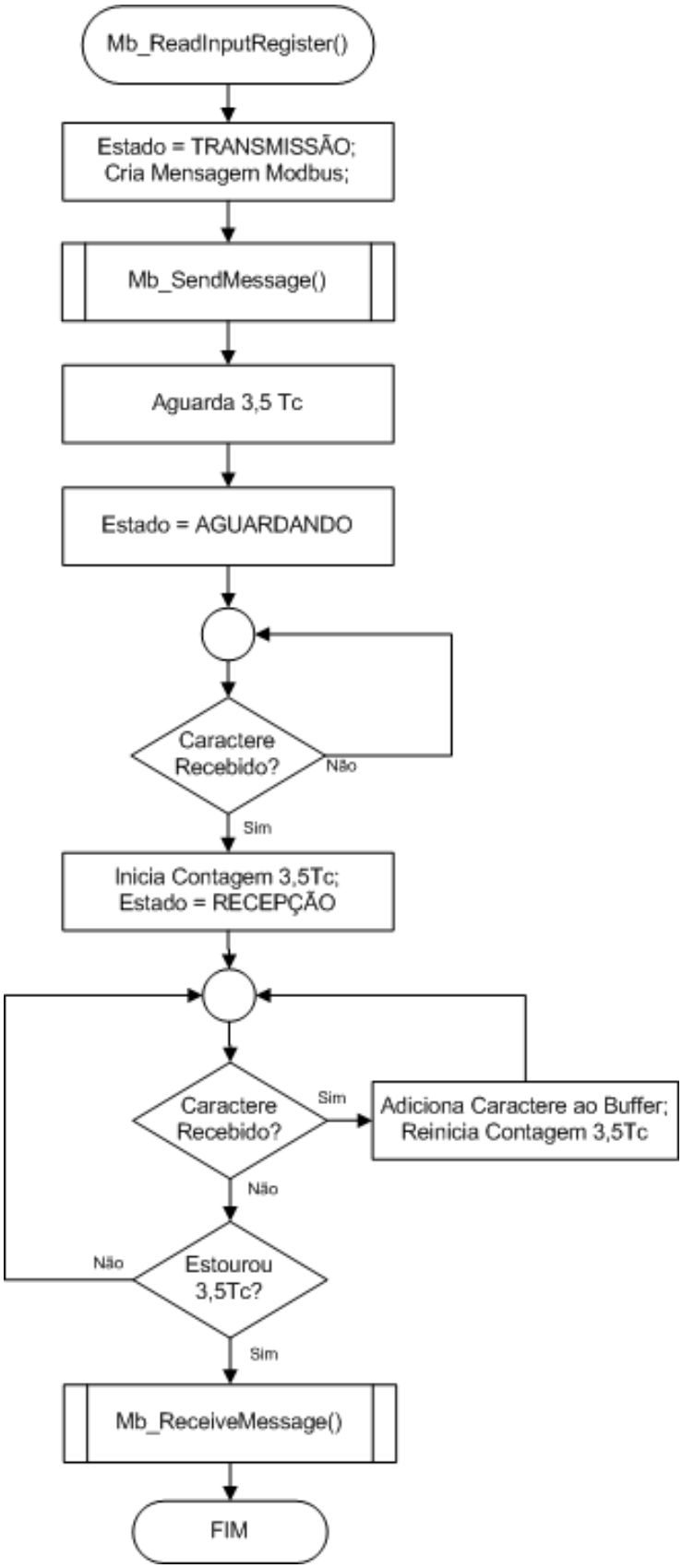
```
typedef struct {
      unsigned char address;
      unsigned char cmd;
      unsigned char data[252];
      unsigned short crc;
      unsigned char dataSize;
} MbRTU;
```

A partir desta estrutura, uma variável global com nome **mbFrame** foi criada para gerenciar a troca de mensagens Modbus na aplicação. Nesta camada de aquisição de dados foram implementadas as funções descritas na sequência. Maiores detalhes em relação ao código, pode ser visualizados no APÊNDICE A – Código Camada de Aquisição.

3.2.1.2 Função Modbus Read Input Register

Para executar a operação Modbus *Read Input Register*, foi criada a função com o protótipo **mb_ReadInputRegister().** O seu algoritmo pode ser visto na Figura 3-8.

Figura 3-8 - Algoritmo da função Read Input Register



FONTE: O AUTOR (2015)

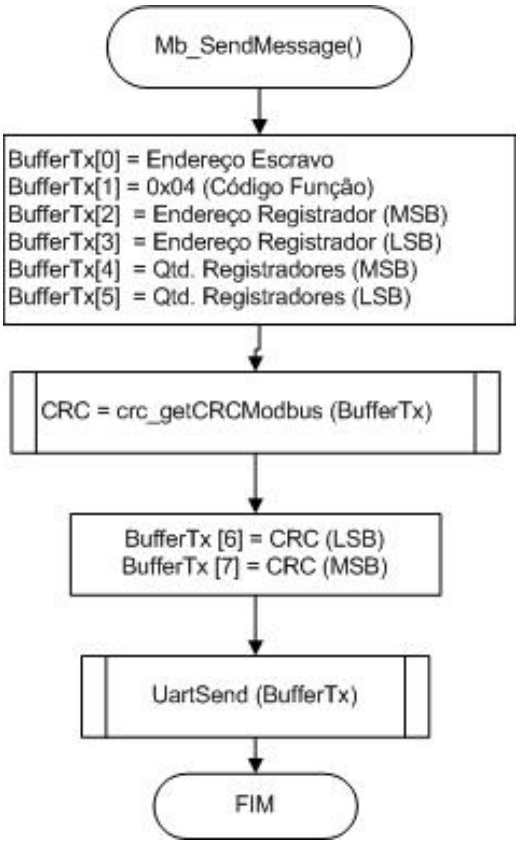
Esta função recebe como parâmetros o endereço do escravo, o endereço do primeiro registrador que deseja realizar a leitura, a quantidade de registradores e o índice referente a identificação deste registrador na memória da Interface Integradora. Foi definido a criação de 4 registradores na memória da interface integradora, para serem utilizados na aquisição.

Os campos da mensagem Modbus são preenchidos de acordo com a Figura 3-3 e enviado pelo mestre ao escravo. Após, o mestre aguarda a recepção da resposta e analisa os dados recebidos.

3.2.1.3 Função Enviar Mensagem Modbus

Dentro da função **`mb_ReadInputRegister(),`** após criar a mensagem Modbus deseja-se enviar a mensagem através da porta serial RS-485. Esta ação é realizada através da função **`Mb_SendMessage().`** O algoritmo desta função pode ser visto na Figura 3-9

Figura 3-9 - Algoritmo da função para enviar mensagem Modbus



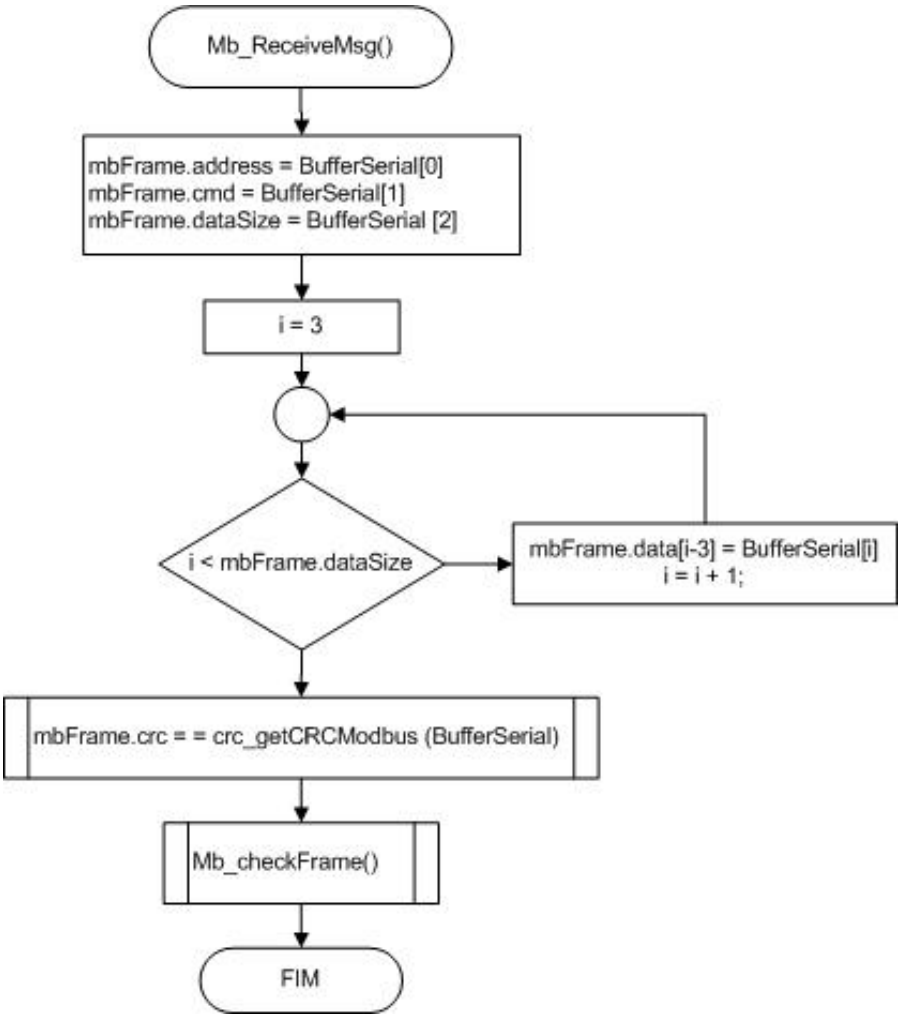
FONTE: O AUTOR (2015)

A função recebe como argumento um ponteiro para a estrutura da mensagem (**mbFrame**). O objetivo desta função é armazenar todos os dados da mensagem em um vetor chamado **BufferTX** que será enviado como argumento para a função interna do microcontrolador **UartSend(),** responsável por efetivamente transmitir os dados pela porta serial.

3.2.1.4 Função Receber Mensagem Modbus

Dentro da função **mb_ReadInputRegister()**, quando deseja-se receber uma mensagem Modbus através da porta serial RS-485 é chamada a função **Mb_ReceiveMessage()**. O algoritmo desta função pode ser visto na Figura 3-10.

Figura 3-10 - Algoritmo da Função Receber Mensagem Modbus



FONTE: O AUTOR (2015)

Esta função armazena na estrutura **mbFrame** os dados que foram recebidos no buffer da porta serial, correspondente a resposta do escravo. Após o recebimento, processa a verificação destes dados.

3.2.1.5 Função para Verificação dos Dados

Após o recebimento da resposta do escravo, a função **mbCheckFrame()** é responsável por verificar os dados recebidos. Esta função verifica se o endereço de destino e o código da função correspondem aos valores enviados pelo mestre. Além disso, verifica a integridade dos dados, fazendo o cálculo do CRC da mensagem e comparando com o valor de CRC recebido.

3.3 Implementação da Camada de Processamento

Na camada de processamento, deve ser implementado toda a lógica de processamento da Interface Integradora. Ela deve ser responsável por gerenciar todas as configurações e parâmetros necessários para o funcionamento adequado do sistema, além de controlar as requisições para aquisição de dados desejadas e a disponibilização dos dados através do servidor.

Além disso, esta camada também será responsável pela configuração dos parâmetros de funcionamento do sistema. Será desenvolvido um software configurador que permitirá que sejam personalizados os modos de aquisição, os endereços de leitura Modbus e os canais de leitura dos sinais digitais e analógicos.

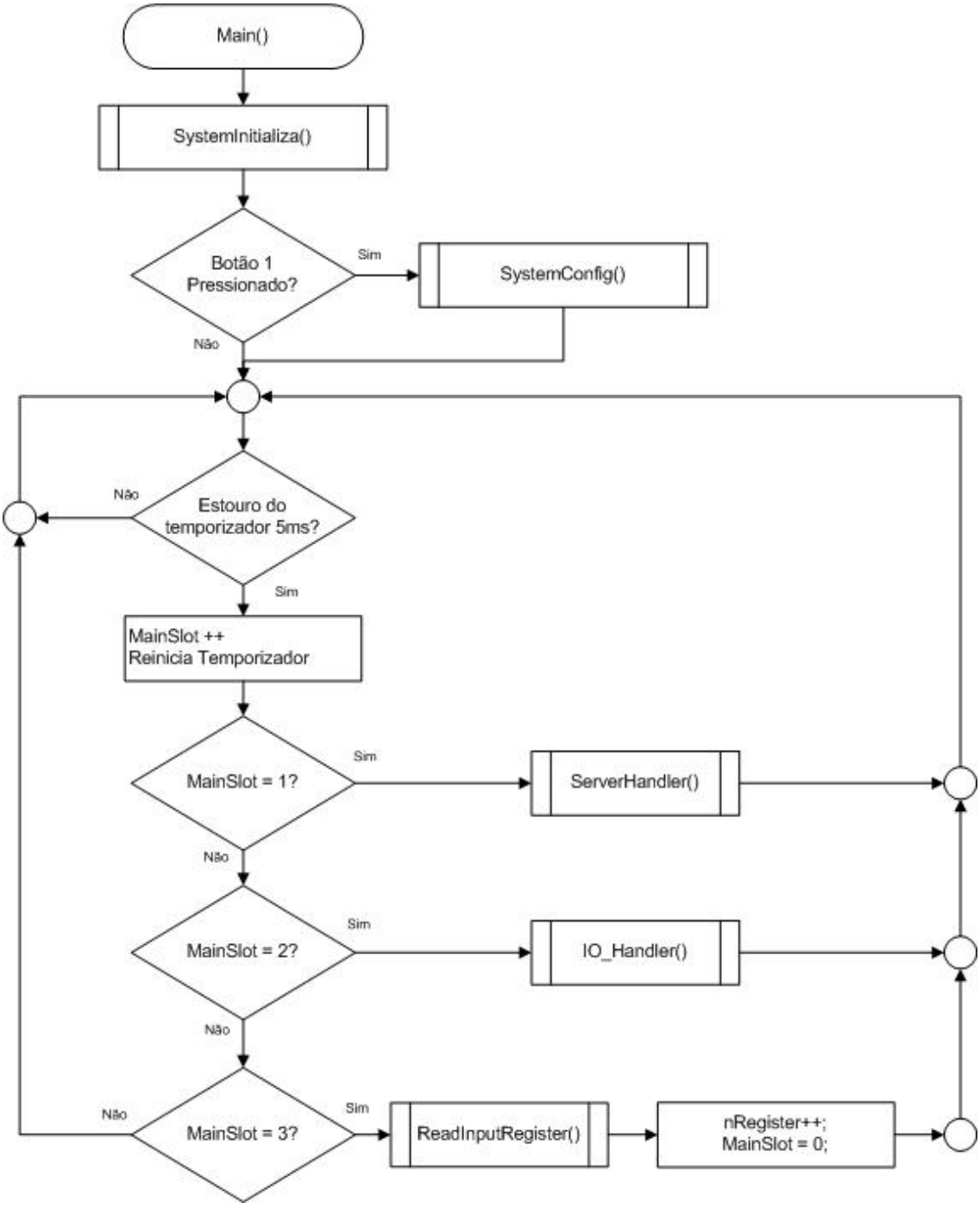
3.3.1 Implementação do Software da Camada de Processamento

O gerenciamento do funcionamento da interface integradora é implementado através da função **main()** como mostra o algoritmo da Figura 3-11. Este é o processo principal do sistema.

Primeiramente, esta função faz a inicialização da aplicação através da rotina **SystemInitialize().** Após inicializada a aplicação, implementa um temporizador com base de tempo de 5ms, juntamente com um contador denominado **mainSlot**. Cada slot define a execução de uma tarefa da interface integradora. Assim a cada

estouro de tempo do temporizador de 5ms, a variável **`mainSlot`** é incrementada definindo qual processo, qual o número do slot, será executado. Isto garante o determinismo nas tarefas executadas e um controle preciso do processamento. Quando executar a ultima tarefa, o **`mainSlot`** é zerado para executar novamente a primeira tarefa, repetindo continuamente.

Figura 3-11 - Algoritmo da função principal



FONTE: O AUTOR (2015)

A função **main()** contém três slots, executando três tarefas: Servidor de Dados, Aquisição de Dados através de I/O e Aquisição de Dados Modbus. As três tarefas fazem as chamadas para o seu *handler* específico. Em cada uma destas tarefas, o sistema verifica os canais configurados, realiza a aquisição dos dados de acordo com a configuração, verifica se algum cliente está solicitando os dados e, em caso afirmativo, envia os mesmos através do protocolo Ethernet TCP/IP. Maiores detalhes em relação ao código relativo a esta camada, pode ser visualizados no APÊNDICE B – Código Camada de Processamento

3.3.1.1 Função de Inicialização do Sistema

Na função **SystemInitialize()** é realizado a inicialização do sistema como: Inicialização das portas seriais para comunicação serial RS-485 através da UART3, inicio dos temporizadores, inicio das funções Modbus através do **Mb_Init()** e estabelecimento de conexão do servidor através da função **TCP_Init().**

3.3.1.2 Software Configurador

O software configurador desenvolvido pode ser executado em qualquer computador que utilize o sistema operacional Windows. A conexão entre o computador e a interface gráfica deve ser feita através da rede local, ou ponto-a-ponto através da porta Ethernet. Através dele é possível selecionar quais os modos de aquisição serão utilizados, configurar os endereços Modbus dos 4 registradores de leitura e os canais desejados para adquirir os sinais digitais e analógicos.

A interface gráfica do software configurador desenvolvido pode ser visto na Figura 3-12. Na parte superior, os campos de IP e porta para acesso à interface integradora devem ser preenchidos de acordo com os dados da interface que deseja acessar. Abaixo destas, estão localizadas as configurações de aquisição de dados.

Para realizar a configuração da Interface Integradora, primeiramente o usuário deve energizar a placa mantendo o botão "1" pressionado. Assim, o sistema irá entrar em modo de configuração, indicando no LCD da placa a mensagem "Modo Configuração". Para retornar ao modo de operação normal deve ser pressionado o botão "2". Quando em modo configuração, a Interface Integradora opera como

servidor, aguardando as solicitações do software configurador que opera como cliente.

Figura 3-12 - Software Configurador da Interface Integradora

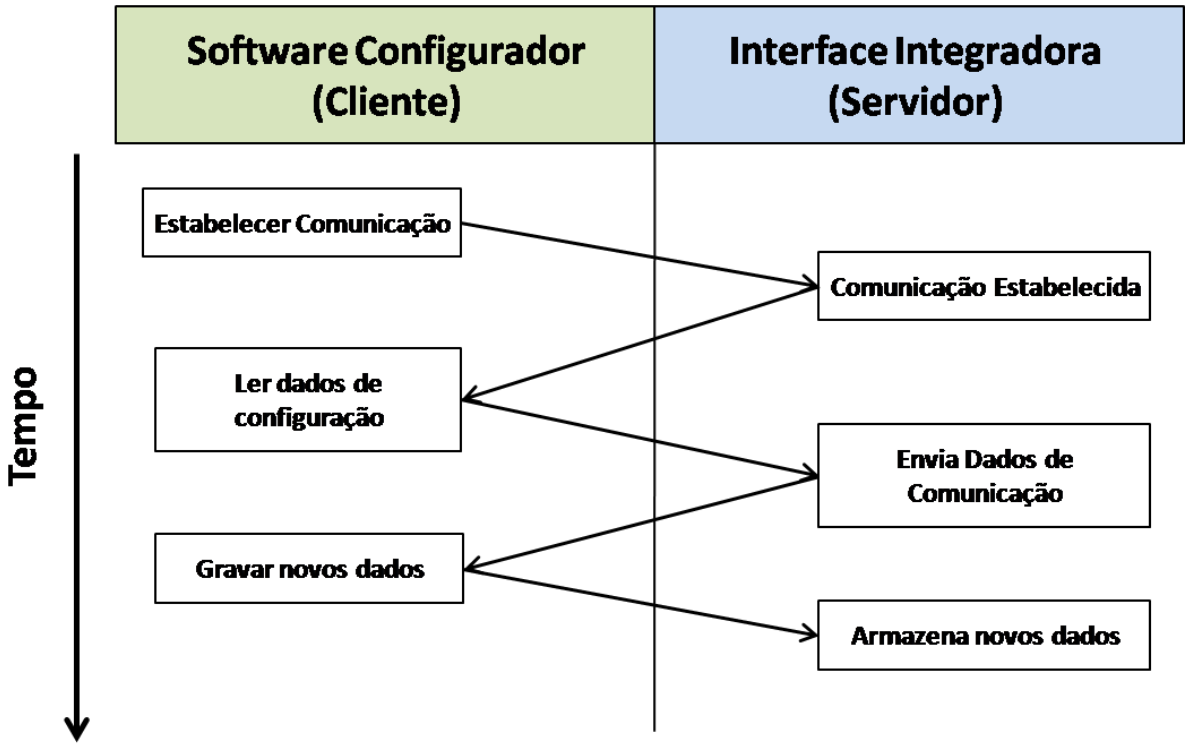
FONTE: O AUTOR (2015)

Do ponto de vista do servidor, a configuração é gerenciada pelo **`SystemConfig(),`** que continuamente faz a chamada da função **`ConfigHandler().`** Esta função implementa três comandos que respondem as solicitações do configurador: Estabelecer conexão, carregar dados e gravar dados. A sequência correta de solicitações implementadas no software configurador e as respectivas respostas da interface integradora, para correta configuração dos parâmetros, pode ser vista na Figura 3-13.

Deste modo, primeiramente o software configurador faz a solicitação para iniciar a comunicação através do IP e porta. O servidor responde com a confirmação de que a comunicação foi estabelecida. Após receber a confirmação, o software faz

a solicitação dos dados armazenados na interface integradora. Recebido os dados, o usuário pode alterar livremente todos os parâmetros, devendo selecionar a opção "Gravar" na parte inferior da tela, para que dados editados sejam enviados a interface e gravados na memória.

Figura 3-13 - Interação entre software configurador e interface integradora para configuração



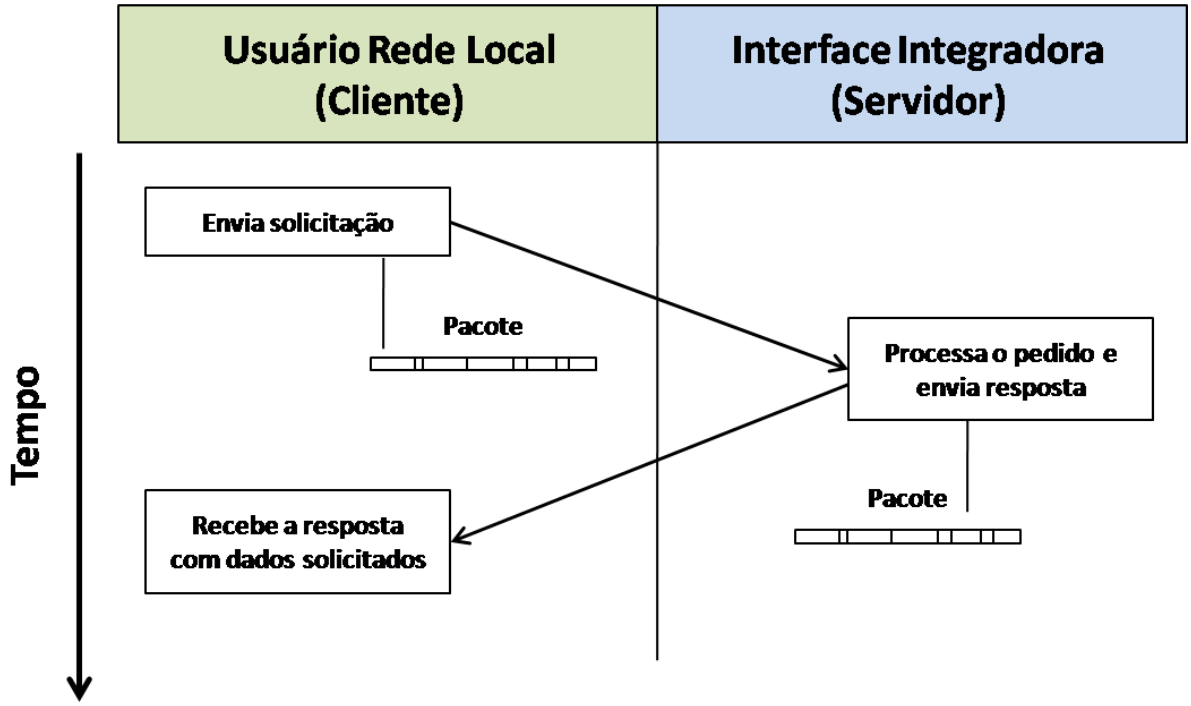
FONTE: O AUTOR (2015)

Após finalizar a configuração, ao pressionar o botão "2" o sistema vai para o modo de operação e passa a funcionar com os novos parâmetros.

3.4 Camada de Servidor de Dados

Na camada de Servidor de Dados, os dados já adquiridos e processados pela camada de processamento, devem ser enviados pela interface integradora até um cliente da rede local, quando os dados forem solicitados, como mostra a Figura 3-14.

Figura 3-14 - Comunicação entre cliente e servidor através de protocolo Ethernet TCP/IP



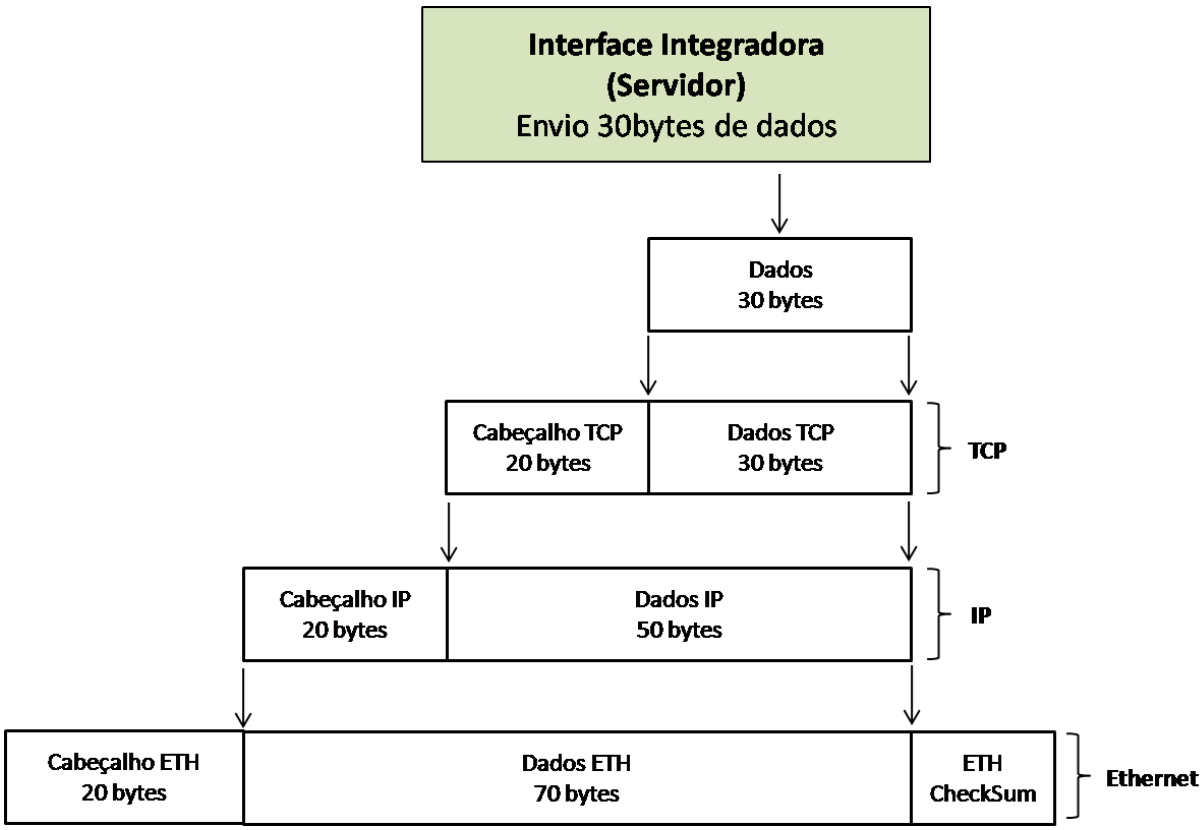
FONTE: O AUTOR (2015)

A conexão entre a Interface Integradora e os clientes da rede local é realizada através de um cabo de rede Ethernet. Os dados que estão na memória da Interface Integradora serão enviados através de protocolo de comunicação Ethernet TCP/IP. O modo como os dados são transmitidos através deste protocolo, pode ser verificado no esquema apresentado na Figura 3-15.

Para transmitir os dados, a Interface Integradora deve primeiramente estabelecer uma comunicação com o cliente. Esta tarefa é implementada pelo protocolo TCP. O TCP recebe os dados que devem ser enviados e acrescenta as informações necessárias de conexão, criando um segmento. Este segmento então receberá as informações relativas a transferência na rede, que é implementado pelo protocolo IP. Assim, o IP adiciona as suas informações ao segmento, criando um pacote. Por sua vez, o pacote então deve ser efetivamente transferido através da rede local utilizando o protocolo Ethernet, que também adiciona um cabeçalho com informações específicas do protocolo e transmite os dados no meio físico.

Antes de realizar a implementação, deve-se compreender os aspectos técnicos dessa tecnologia de comunicação.

Figura 3-15 - Envio de dados para rede local através de Ethernet TCP/IP



FONTE: O AUTOR (2015)

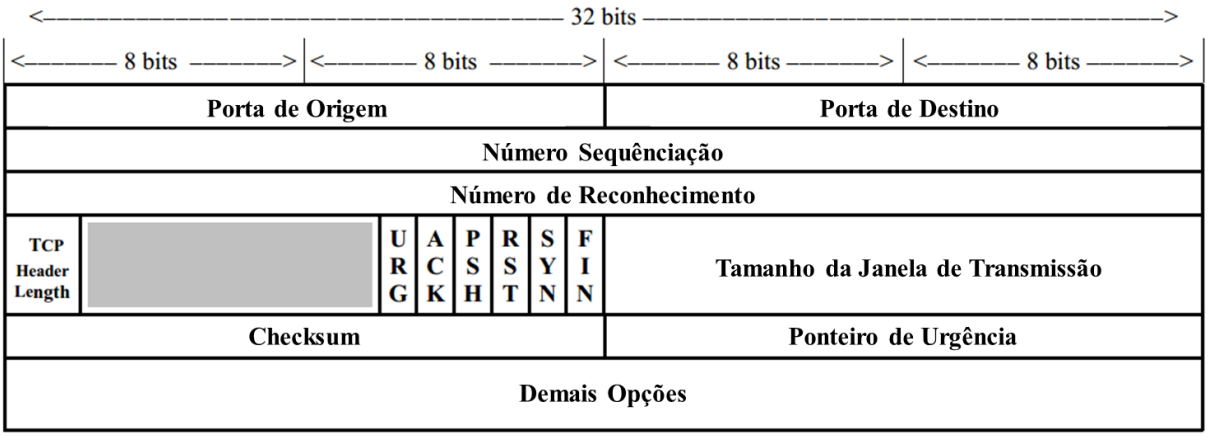
3.4.1 Protocolo TCP

Transmission Control Protocol, conhecido pela abreviação TCP, é um protocolo que estabelece uma ligação entre dois dispositivos, semelhante à maneira que um telefone estabelece uma ligação entre duas pessoas. As conexões TCP são full-duplex, o que significa que os dados podem ser enviados e recebidos ao mesmo tempo. O TCP é um protocolo de dados por pacotes,ou seja, ele divide os dados de entrada em pequenos pacotes chamados segmentos. O tamanho máximo de um segmento TCP é ajustável, mas deve ser inferior a 65536 bytes. O TCP adiciona um cabeçalho de 20 bytes no início de cada um dos segmentos (ver Figura 3-16), em seguida, passa este segmento TCP para a camada do protocolo IP para transmissão (METCALFE, 1976).

Antes de estar apta a comunicar-se usando TCP, a interface integradora deve primeiramente estabelecer uma comunicação. Assim, primeiramente deve ser criado

um *socket* TCP nela e no servidor da rede local. Um *socket* é o ponto final de uma conexão TCP e consiste em um endereço IP e um número de porta. Normalmente, um aplicativo de servidor cria um *socket* de escuta, responsável por aguardar que um cliente entre em contato solicitando informações. Portanto, a interface integradora deve criar um *socket* e conecta-lo ao *socket* de escuta do servidor.

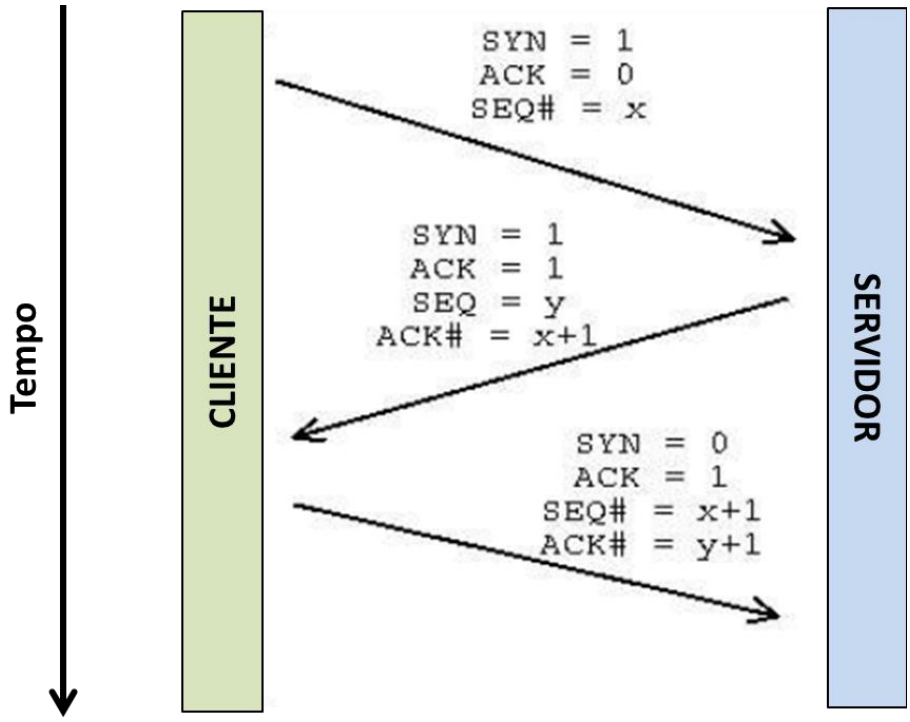
Figura 3-16 - Segmento do protocolo TCP



FONTE: Adaptado de Tanenbaum (1999).

Uma vez que os sockets foram criados, a conexão deve ser estabelecida entre os dispositivos através de um mecanismo conhecido como *handshake* (aperto de mão em tradução livre). Esse mecanismo é conhecido como um processo de três vias, conforme mostra a Figura 3-17 - Mecanismo *Handshake* para estabelecer comunicação entre cliente e servidor .No primeiro passo, o *socket* do cliente envia um segmento TCP para o servidor com os bits de controle “SYN” = 1, “ACK” = 0, e um número qualquer (‘*X*’ por exemplo) no campo de sequencia. No segundo passo, se o servidor estiver na escuta e aceitar a conexão, ele responde ao cliente com um segmento TCP que contém os bits de controle “SYN” = 1, ACK = 1, o número de reconhecimento definido como‘*X*+1’ e o número da seqüência definido como um número qualquer (“*Y*” por exemplo). Por último, o cliente deve confirmar a conexão com um segmento TCP contendo “SYN” = 0, “ACK” = 1, número de reconhecimento = ‘*Y+1’* e número da sequência = ‘X+1’. Assim, a conexão esta estabelecida e ambos os dispositivos estão prontos para a troca de dados, que deverá ser implementada através do protocolo IP ((METCALFE, 1976).

Figura 3-17 - Mecanismo *Handshake* para estabelecer comunicação entre cliente e servidor



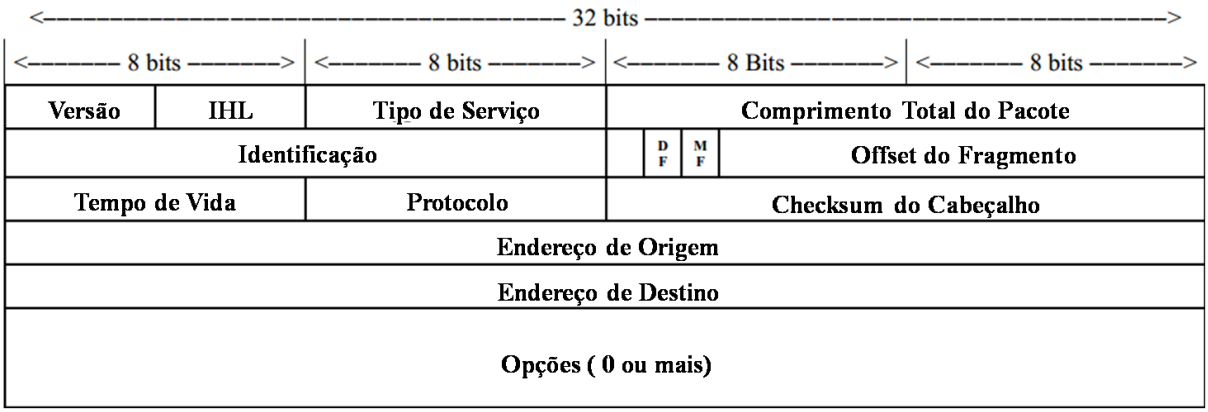
FONTE: O AUTOR (2015)

3.4.2 Protocolo IP

O protocolo IP*(Internet Protocol)* é responsável pela transferência de dados através da rede. Da mesma maneira que o TCP, o IP é um protocolo baseado em pacotes que, recebe os dados em segmentos e adiciona a cada um deles um cabeçalho característico, formando um pacote maior chamado pacote IP. A norma RFC 791 descreve todos os campos do pacote e suas funcionalidades, sendo o tamanho máximo de 65535 bytes com um cabeçalho composto por 20 bytes como mostra a Figura 3-18. Para os objetivos do trabalho, os campos mais importantes são: protocolo, endereço de origem e endereço de destino.

Os endereços IP de origem e destino, identificam quem enviou o pacote e quem deve receber. Já o campo de protocolo identifica qual o protocolo que está fornecendo os dados para serem acrescidos os campos do protocolo IP. No caso da interface integradora, o protocolo utilizado é o TCP e este é identificado pelo número 6.

Figura 3-18 - Pacote IP



FONTE: Adaptado de Tanenbaum (1999).

Além do TCP, existe também dentro do protocolo IP um outro protocolo chamado ICMP, definido pela norma RFC 792, e utilizado para troca de mensagens relativas a erros ocorridos na transmissão. Qualquer estação que utilize IP, precisa identificar e processar as mensagens ICMP e alterar o seu comportamento de acordo com o erro relatado. Os *gateways*, roteadores e placas de rede, são programados de fábrica para troca de mensagens ICMP quando receberem pacotes que provoquem algum erro. Os erros mais comuns que ocorrem são:

- Um pacote IP não consegue chegar ao seu destino;

- Barramento congestionado;

- O Roteador encontrou uma rota mais rápida para o envio dos pacotes.

Finalmente, depois de criado o pacote IP, o mesmo deve ser transferido para a rede local através do protocolo Ethernet.

3.4.3 Rede Local - Ethernet

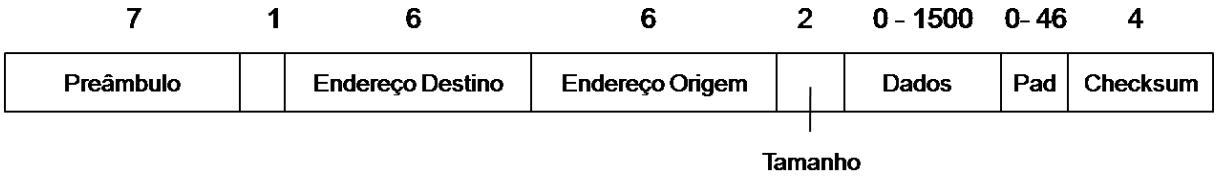
Ethernet é um padrão para o transporte de pacotes de dados digitais entre estações conectadas a uma rede local. O mecanismo de transporte de pacotes fornecido pelo Ethernet é a tecnologia de rede local mais utilizada no mundo. Seu desenvolvimento iniciou em 1973, dentro da Xerox Corporation. Em 1976, foi construída a primeira rede de 2,94 Mbps para conectar mais de 100 estações de trabalho pessoais em um cabo de 1 km. Esta rede foi chamada de Ethernet e foi

utilizada como uma base para a especificação IEEE 802.3 em 1985. "Ethernet" refere-se a um produto que antecede o padrão IEEE 802.3. No entanto, atualmente qualquer rede compatível 802.3 é definida como Ethernet.

Ao longo dos anos este padrão continuou a evoluir. Em 1995, foi desenvolvido o Ethernet 100 Mbps aumentando a velocidade de comunicação, que desde então tem sido incrementada com o lançamento do Gigabit Ethernet e, recentemente, 100 Gigabit Ethernet (100 GbE) que trabalha com velocidade de 100Gbps. No futuro, a Ethernet irá continuar aumentando em velocidade e a perspectiva é de predominar esta tecnologia para comunicação de dados.

A estrutura da mensagem do protocolo Ethernet é conhecida como *frame* cujos campos estão organizados conforme mostra a Figura 3-19.

Figura 3-19 - Quadro do Protocolo Ethernet



FONTE: Adaptado de Tanenbaum (1999).

Preâmbulo - Este campo é composto por sete bytes com valor fixo na forma binária ‘101010....’. O motivo para esta padronização é permitir a sincronização do clock entre receptor e emissor.

Delimitador - Este é o oitavo byte do frame, e possui valor fixo na forma binária ‘10101011’. A diferença desta para o preâmbulo está no último bit, que justifica a função deste campo, que é usado para separar os bytes de sincronização do resto do *frame.*

Endereço Destino - Endereço do destinatário pretendido pelo *frame*. Os endereços no padrão 802.3 utilizam endereços únicos globais de 48 bits.

Endereço Origem - Endereço da fonte, da mesma forma que de destino.

Tamanho - Indica o tamanho do campo de dados do *frame* Ethernet, que pode ser de 0 até 1500 bytes.

Dados - Esta é a informação a ser enviada pelo *frame*.

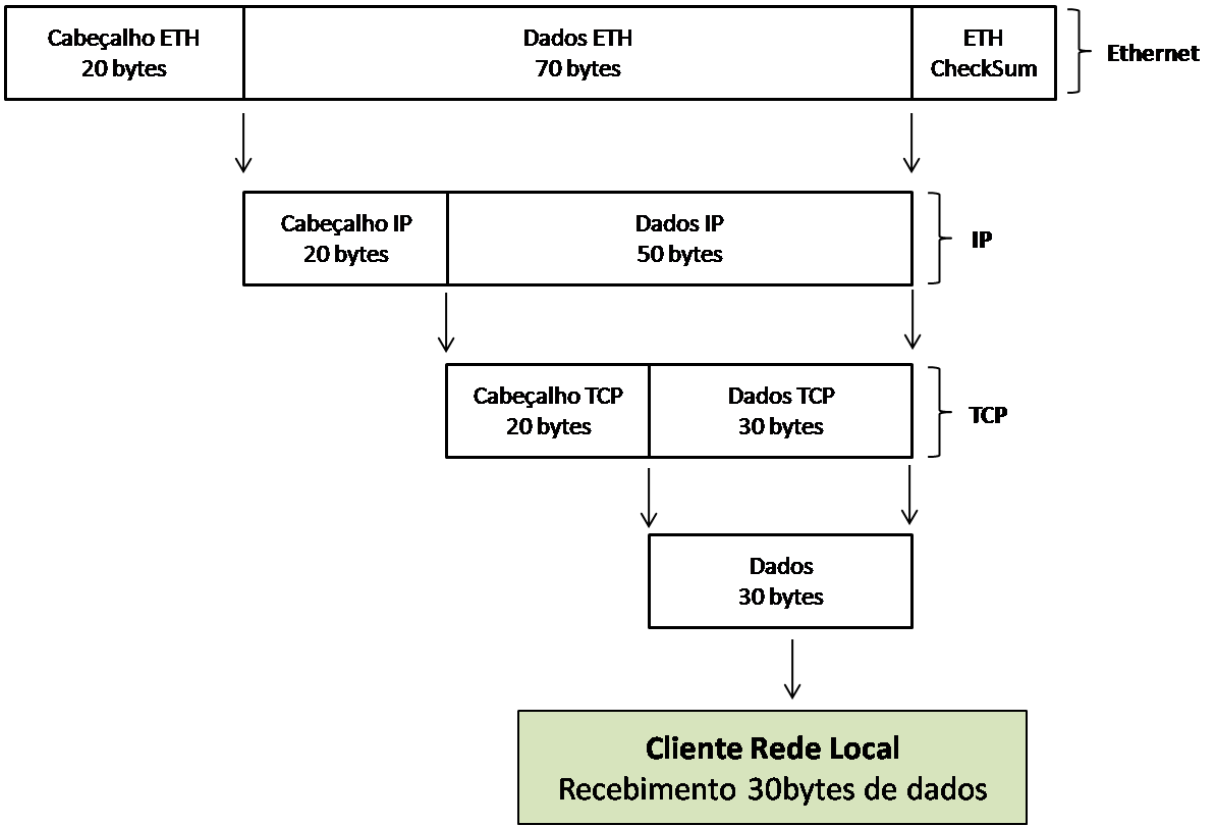
Pad - Apesar de ser permitido que o campo de dados tenha tamanho zero, o protocolo Ethernet especifica que deve haver pelo menos 64 bytes em todo o *frame.*Como todos os campos somados, considerando que o campo de dados esteja vazio, chega-se no máximo a 18 bytes. Assim, foi adicionado o campo 'Pad' composto por 46 bytes de dados com o objetivo de preencher o *frame* para atingir o número mínimo de bytes requeridos.

Cheksum - verificação de redundância cíclica (CRC) para detectar os erros que ocorrem durante a transmissão ou **FCS** Frame Check Sequence.

Finalizado o processo de montagem do pacote completo, desde os dados que a interface integradora deseja enviar, passando pela criação dos pacotes do protocolo TCP, do pacote IP e do Pacote Ethernet, obtem-se um pacote pronto para ser enviado a estação cliente, denominado Ethernet TCP/IP.

Na estação cliente, o pacote Ethernet TCP/IP passa pelo processo inverso como pode ser visto na Figura 3-20.

Figura 3-20 - Recebimento de Dados através do protocolo TCP/IP



FONTE: O AUTOR (2015)

Começando desta vez pelo protocolo Ethernet, cada camada retira as suas informações do pacote, processando os dados e entregando para a camada superior o restante, até chegar apenas aos dados de interesse para a aplicação do usuário.

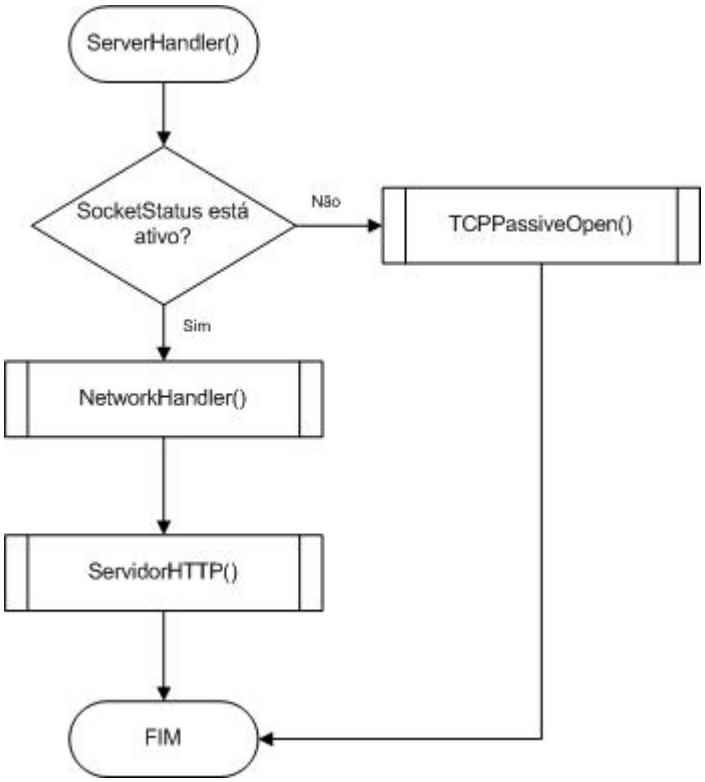
3.4.4 Implementação do Software Camada de Servidor de Dados

A implementação do software desta camada foi realizada utilizando as bibliotecas do uVision fornecidos pelo Keil®. O software de programação possui biblioteca que implementa a pilha do protocolo TCP/IP, realizando as operação de baixo nível do protocolo de acordo com as especificações. Assim, cabe ao programador desenvolver as funções de alto nível, como demonstrado a seguir.

3.4.4.1 Função Gerenciador Servidor

O gerenciamento do servidor na camada de servidor de dados é realizada pela função **`ServerHandler()`**. O algoritmo da função pode ser visto na Figura 3-21.

Figura 3-21 - Algoritmo da função gerenciador do servidor



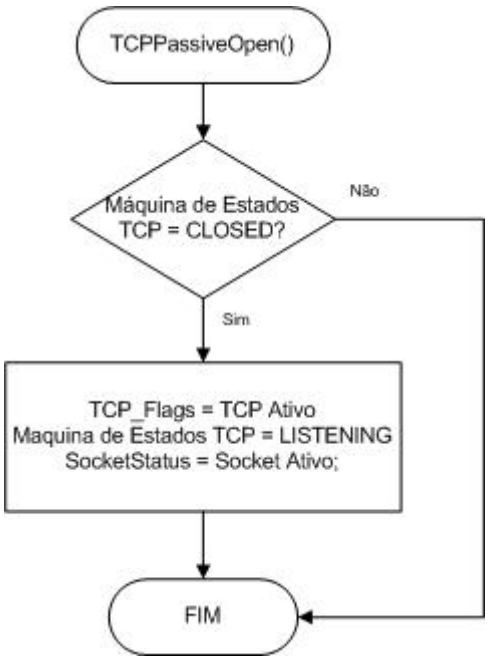
FONTE: O AUTOR (2015)

Está função é continuamente chamada no programa principal e é responsável por verificar a conexão socket entre a interface integradora e o cliente, realizar as operações de baixo nível da rede Ethernet TCP/IP e enviar os dados do servidor para o cliente, quando solicitado.

3.4.4.2 Função Abrir Conexão TCP

O início da comunicação servidor/cliente acontece através da função **`TCPPassiveOpen()`**. O algoritmo desta função pode ser visto na Figura 3-22

Figura 3-22 - Algoritmo da função abrir conexão TCP



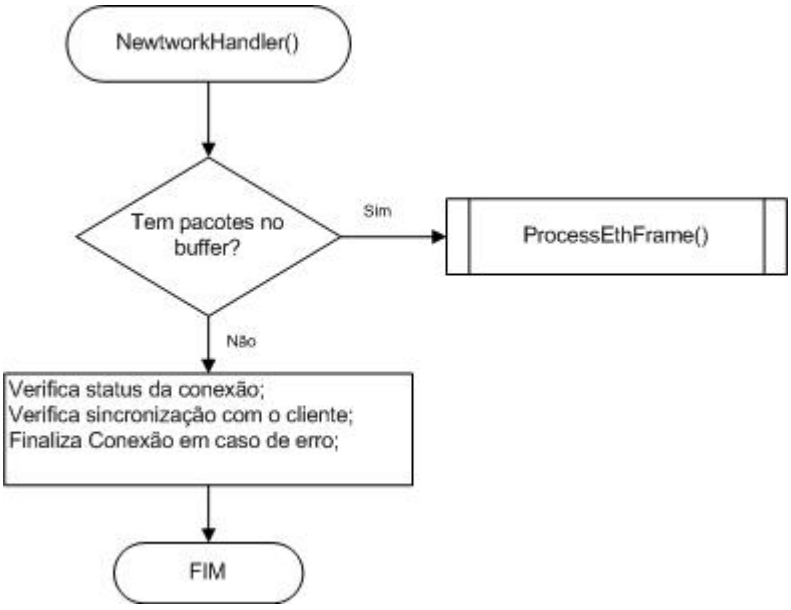
FONTE: O AUTOR (2015)

Está função cria o socket e limpa todas as flags de comunicação, levando também a máquina de estados TCP para **`LISTENING`**, assim as funções de baixo nível do TCP/IP identificam que a conexão está iniciada. Após executar esta função, a interface integradora está apta a receber solicitações de clientes da rede local através da porta configurada.

3.4.4.3 Função Gerenciamento TCP/IP

Iniciada a conexão *socket* na interface integradora, a função **`NetworkHandler()`** gerencia as operações de baixo nível do TCP/IP no microcontrolador. O algoritmo desta função pode ser visto na Figura 3-23.

Figura 3-23 - Algoritmo da função de gerenciamento TCP/IP



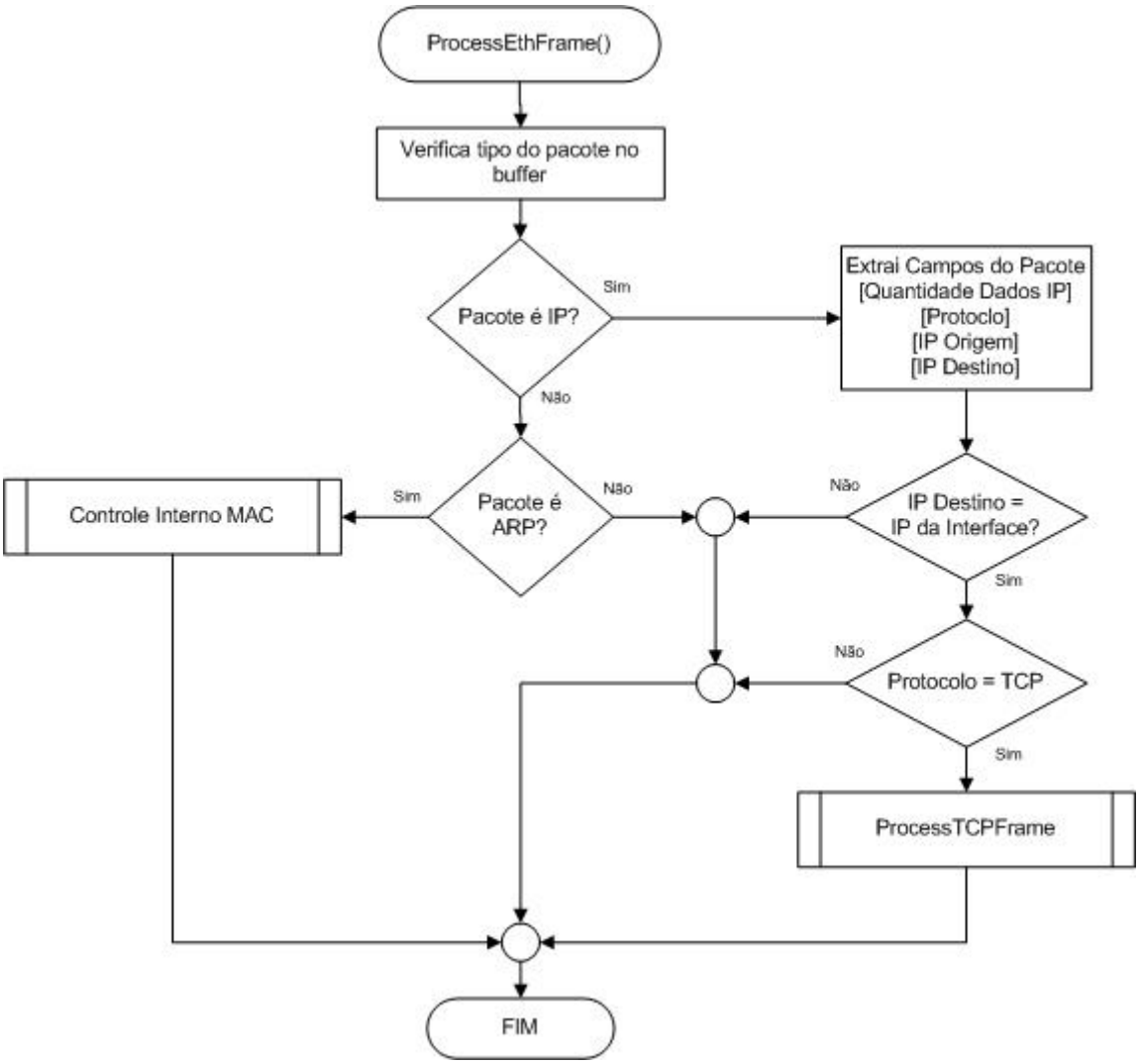
FONTE: O AUTOR (2015)

Esta função monitora os eventos do cliente na rede, indicando através de flags de comunicação o status de recebimento e transmissão, assim como falhas na conexão. Além disso, ao identificar que existe um pacote no buffer de recebimento da porta Ethernet, é realizada uma chamada para a função responsável pelo processamento do pacote.

3.4.4.4 Função Processamento de Pacote Recebido

Ao identificar o recebimento de um pacote na rede, de acordo com o protocolo Ethernet TCP/IP, a interface integradora deve verificar se esse pacote é direcionado a ela e, em caso afirmativo, responder ao cliente com os dados solicitados. Esta tarefa de processamento do pacotes é realizada pela função **`ProcessEthFrame()`** de acordo com o algoritmo da Figura 3-24.

Figura 3-24 - Algoritmo da função de processamento de pacote



FONTE: O AUTOR (2015)

Nesta etapa, é verificado primeiramente se o pacote corresponde ao protocolo ARP, que caracteriza pacotes de identificação de endereços físicos e lógicos dentro da rede. Estes pacotes são gerenciados automaticamente pelo baixo nível do TCP/IP no microcontrolador, assim como pela placa de rede dos computadores e servidores. Portanto, caso os pacotes forem do protocolo ARP, devem ser ignorados pela aplicação.

Entretanto, caso os pacotes implementarem o protocolo IP, indica o envio de um pacote contendo dados relevantes. Desta maneira, a interface integradora interpreta cada um dos campos do pacote. Primeiramente, o endereço de destino é verificado para se certificar que foi enviado para a interface integradora, devendo

então ser processado. Caso contrário, o mesmo deve ser descartado. Após, o protocolo de transporte verifica se o pacote pertence ao protocolo ICMP ou TCP. No caso do ICPM, deve ser utilizada a função da biblioteca TCP/IP chamada de **`ProcessICMPFrame()`**. No caso de TCP, quem realiza o processamento é a função **`ProcessTCPFrame()`**. Após a execução desta função, os dados recebidos serão armazenados no buffer de comunicação e as flags de recebimentos de dados serão alteradas para indicar que existem solicitações dos clientes a serem respondidas.

3.4.4.5 Função Resposta ao Cliente

Ao identificar o recebimento de uma solicitação de um cliente, a resposta é processada e transmitida através da função **`ServerResponse()`** conforme algoritmo da Figura 3-25.

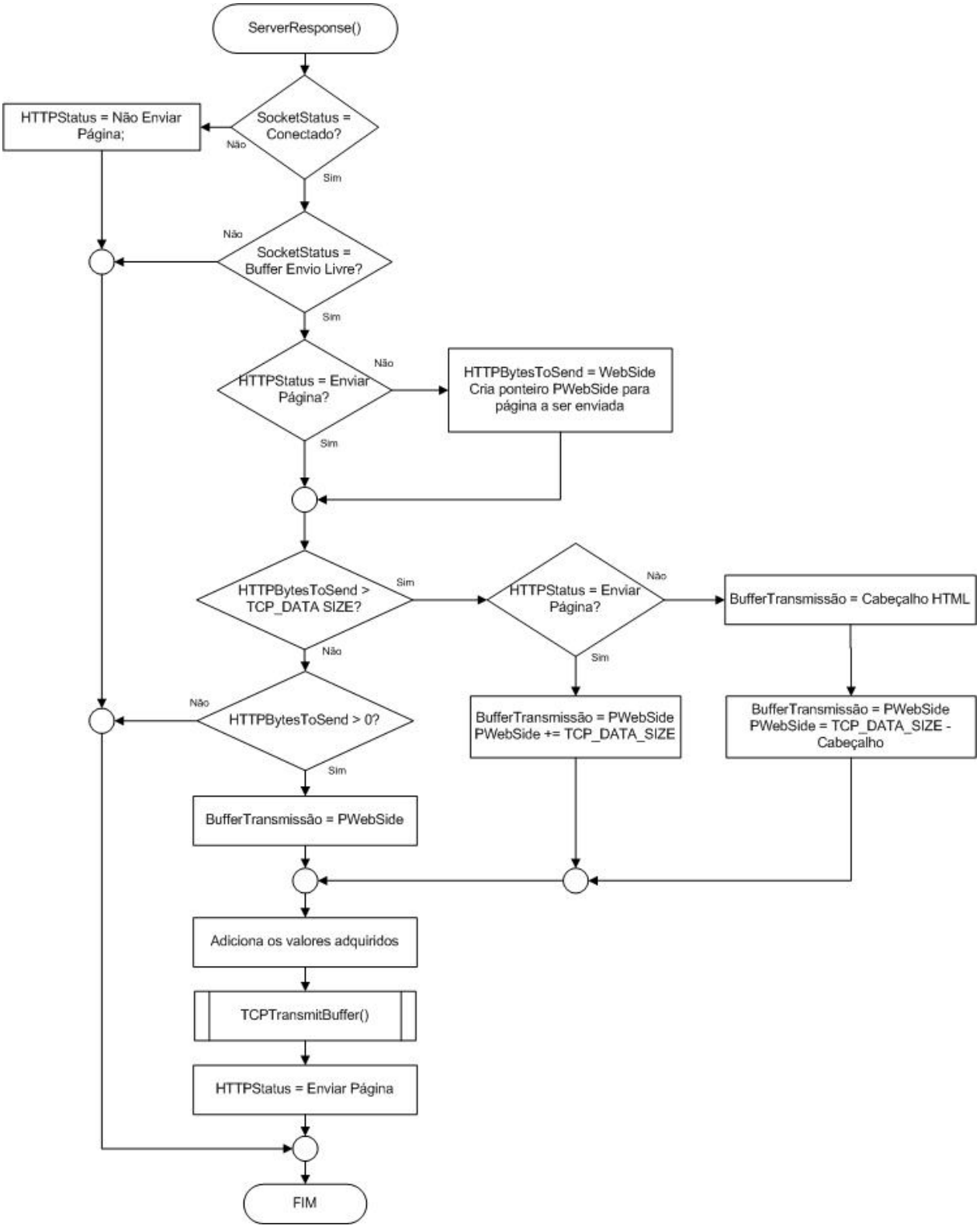
Nesta função primeiramente é verificado o status da conexão TCP através da flag **`socketStatus,`** para garantir que a conexão socket está estabelecida e funcionando normalmente, além de certificar-se que a interface integradora está livre para transmitir os dados.

A mensagem de resposta do servidor de dados consiste em um página HTML com as informações referente aos dados adquiridos. O código HTML desta página fica armazenado na variável **`WebSide.`** Como as informações são transmitidas através do protocolo de aplicação HTTP, primeiramente deve ser enviado o cabeçalho HTTP que fica armazenado na variável **`HTTPHeader.`** Um flag denominada **`HTTPStatus`** indica quando o cabeçalho já foi enviado, autorizando o envio da página HTML ao cliente.

Assim, após as verificações iniciais, caso a flag HTTPStatus não indicar o envio da página, significa que a transmissão não foi iniciada ainda e o cabeçalho HTTP deve ser enviado. Assim é criado um ponteiro para o conteúdo da página HTML, chamada **`PWebside,`** além de contar o número de bytes que deve ser enviado, armazenando este na variável **`HTTPBytesToSend.`** Se o número de bytes for maior que o tamanho máximo de dados permitido em um pacote, a função providencia a divisão dos dados para serem enviados parcialmente, pacote por pacote, até transmitir todos os dados. Esse gerenciamento é feito através da

contagem dos bytes a enviar (**HTTPBytesToSend**) juntamente com o deslocamento do ponteiro **PwebSide.**

Figura 3-25 - Algoritmo da função de resposta ao cliente



FONTE: O AUTOR (2015)

Verificado os dados que devem ser enviados, os mesmos são armazenados na variável global **`TCP_TX_BUF`**, para então ser chamada a função **`TCPTransmittBuffer()`** que irá criar o pacote Ethernet TCP/IP e enviar os dados ao cliente. No final a flag **`HTTPStatus`** é alterada para indicar que a página está sendo enviada.

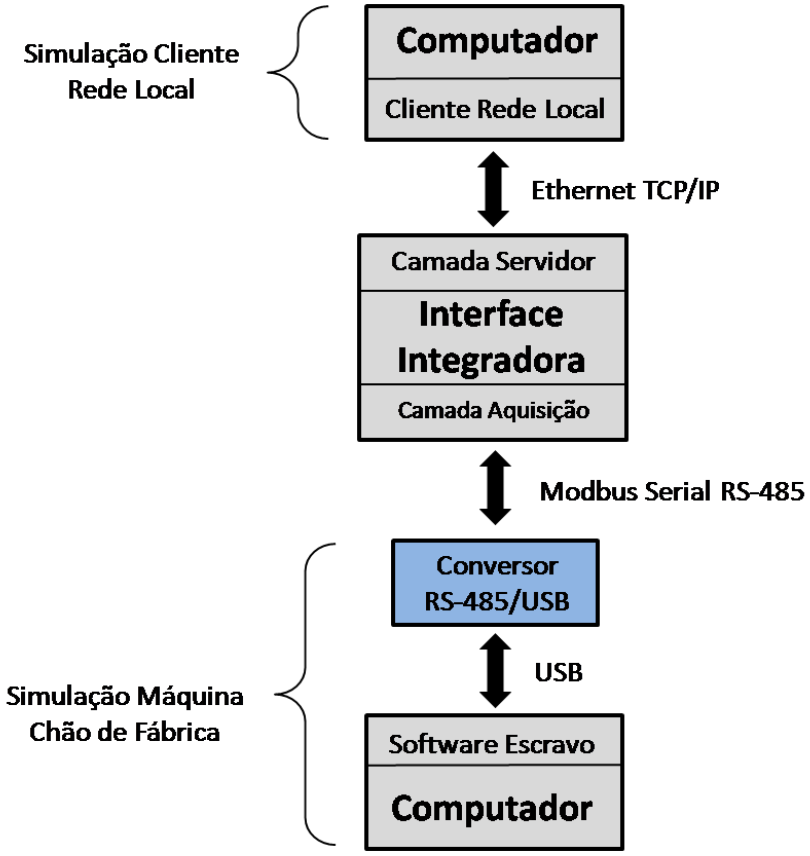
4 RESULTADOS FINAIS

Após a implementação da interface integradora, deve ser realizado simulações do funcionamento para testar as camadas da aplicação antes do teste em ambiente real.

4.1 Simulação da Interface Integradora

Para testar o funcionamento da Interface Integradora, será necessário simular uma máquina operando e fornecendo dados através do Modbus Serial RS-485, além de conectar um cliente em uma rede local junto com a interface integradora, para visualização de dados. Para esta finalidade, será utilizado um computador para ambas as funções, como mostra a Figura 4-1

Figura 4-1 - Esquema de simulação da Interface Integradora utilizando um computador



FONTE: O AUTOR (2015)

Para simular uma máquina, será utilizado um software desenvolvido na plataforma LabVIEW que possui uma biblioteca oficial Modbus implementada no padrão especificado pelo protocolo, sendo assim uma excelente forma para testar a aquisição de dados da interface integradora (ver Figura 4-2). Como o PC não possui suporte ao meio físico RS-485, foi utilizado um conversor RS-485/USB para possibilitar a conexão do PC na rede Modbus Serial RS-485. Ao conectar o conversor na porta USB do computador, o sistema operacional atribui ao dispositivo uma porta COM.

Para a simulação, foram definidos como exemplo 4 registradores para leitura de dados de baixo nível, possuindo estes os endereços 0, 1, 2 e 3 respectivamente. O software do escravo fica continuamente verificando o buffer da porta serial do computador a esperando uma mensagem de um mestre Modbus. Ao receber a solicitação, o software envia uma resposta, com os dados dos registradores solicitados.

Figura 4-2 - Software simulação escravo Modbus

FONTE: O AUTOR (2015)

O software possui uma interface gráfica que possibilita selecionar a porta serial COM do PC a qual estará conectada a rede Modbus, além do timeout de erro de comunicação e o endereço do escravo que deseja-se simular. Na parte inferior, foram criado 4 controles para os registradores, sendo possível alterar o valor de cada registrador em tempo de execução.

Com o conversor conectado na porta COM5, foi selecionado um timeout de 10 segundos e endereço de escravo 3, configurando os registradores com valores 4, 400, 4000 e 40000 respectivamente. Na interface integradora, foi realizada a configuração para realizar a leitura destes 4 registradores em sequência, iniciando pelo registrador de endereço 0. Deste modo, espera-se que a troca de mensagens entre os dispositivos ocorra conforme mostra a Figura 4-3.

Figura 4-3 - Troca de mensagens esperada para comunicação Modbus entre dispositivos

Requisição (Mestre)		**Resposta (Escravo)**	
Campo Mensagem	**Valor (Decimal)**	**Campo Mensagem**	**Valor (Decimal)**
Endereço Escravo	3	Endereço Escravo	3
Código Função	4	Código Função	4
Endereço Inicio MSB	0	Número Bytes	8
Endereço Inicio LSB	0	Valor Registrador 0 MSB	0
Quantidade Registradores MSB	0	Valor Registrador 0 LSB	4
Quantidade Registradores LSB	4	Valor Registrador 1 MSB	1
CRC LSB	67	Valor Registrador 1 LSB	144
CRC MSB	8	Valor Registrador 2 MSB	15
		Valor Registrador 2 LSB	160
		Valor Registrador 3 MSB	156
		Valor Registrador 3 LSB	64
		CRC LSB	192
		CRC MSB	127

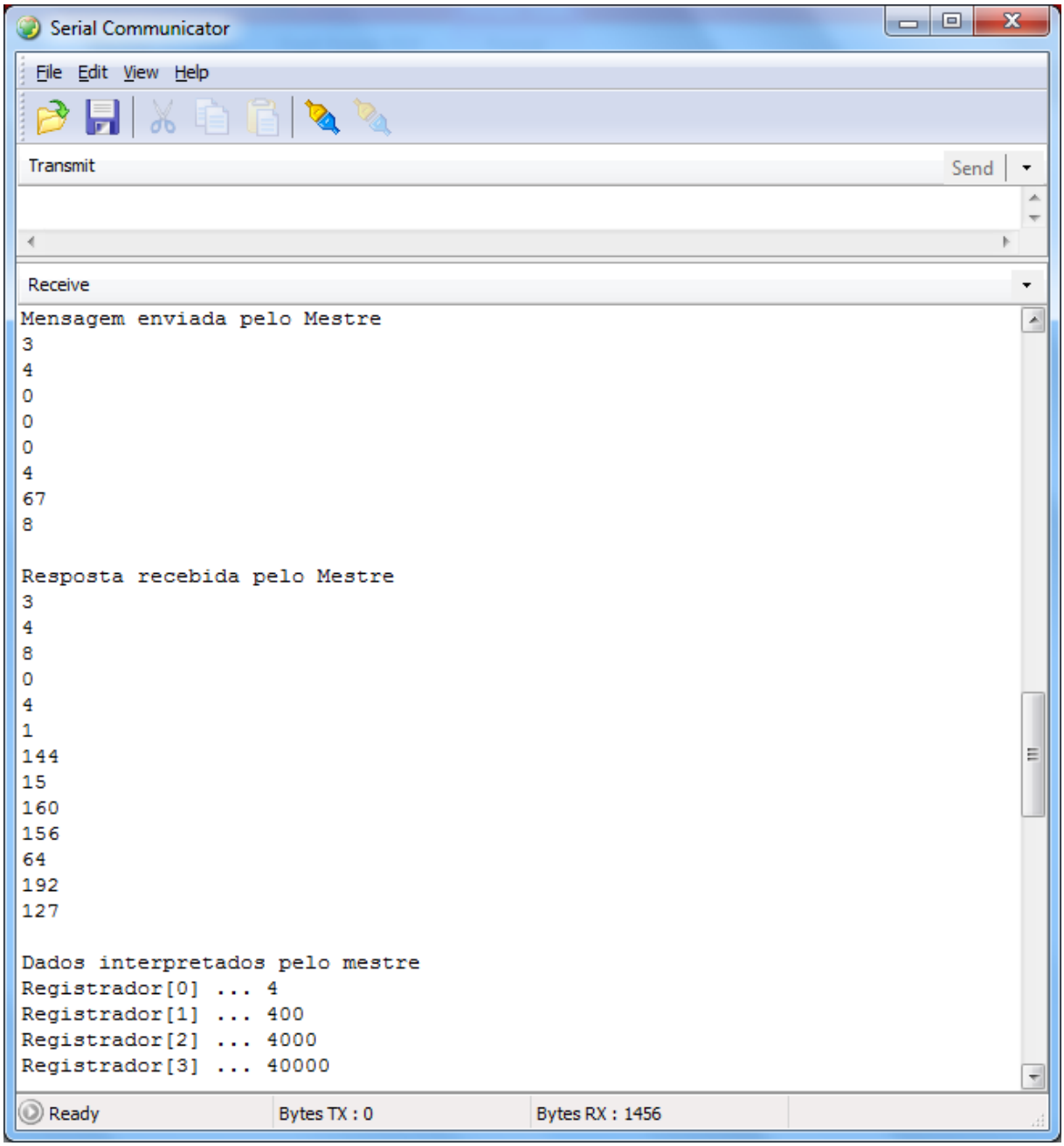
Requisição	**Resposta**
Read Input Register	Read Input Register
Endereço Escravo = 3	Endereço Escravo = 3
Endereço Registrador Inicio = 0	Número Bytes da Resposta = 8
Quantidade Registradores = 4	Valor Registrador (0) = 0x0004 = 4
	Valor Registrador (1) = 0x0190= 400
	Valor Registrador (2) = 0x0FA0 = 4000
	Valor Registrador (3) = 0x9C40 = 40000

Após configurar ambos os dispositivos da rede Modbus, foi iniciado primeiramente o software do escravo no PC e depois energizada a interface integradora. Para facilitar a depuração e visualização da troca de mensagens, a interface integradora foi programada para enviar, através de um porta serial adicional, todas as mensagens enviadas e recebidas na rede Modbus. Assim, através do software “Serial Communicator” é possível monitorar estas mensagens.

Na Figura 4-4, pode-se verificar o resultado da simulação, onde a troca de mensagens corresponde ao esperado previamente na Figura 4-3 .

Ao mesmo tempo em que é simulado a camada de aquisição de dados, pode ser também verificar o funcionamento da camada de Servidor de Dados, conectando a interface entregadora através de um cabo ethernet a placa de rede do mesmo computador. Deste modo, o computador irá operar, tanto simulando um máquina no baixo nível, como o acesso do usuário ao alto nível, conforme mostra a Figura 4-1.

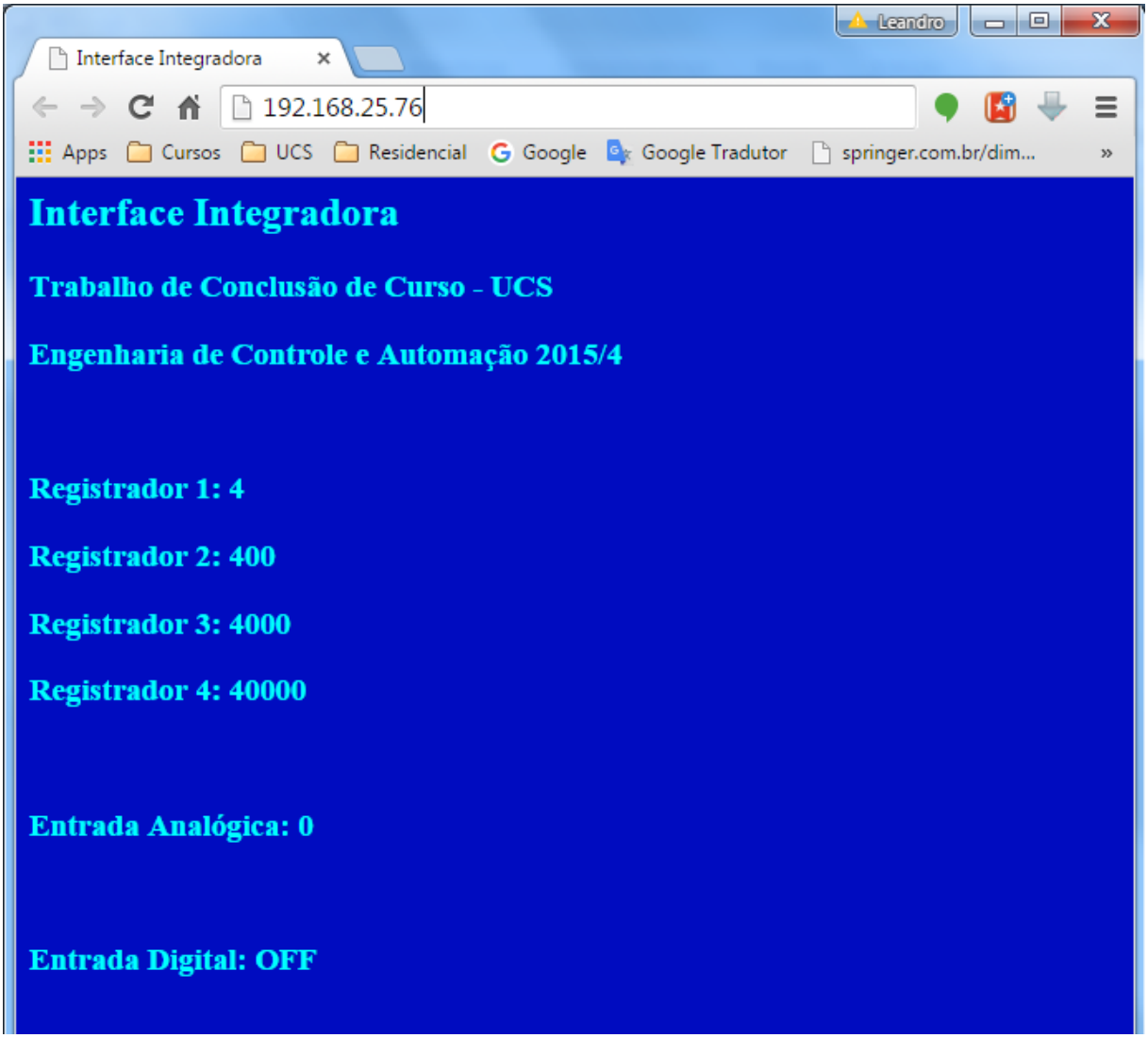
Figura 4-4 - Mensagens da simulação realizada entre Mestre e Escravo



FONTE: O AUTOR (2015)

O IP definido para a interface integradora foi 192.168.25.76. Assim, acessando a partir de um navegador de internet, através do IP configurado, é possível visualizar os dados adquiridos pela camada de aquisição, como mostra a Figura 4-5.

Figura 4-5 - Visualização dos dados através da rede local



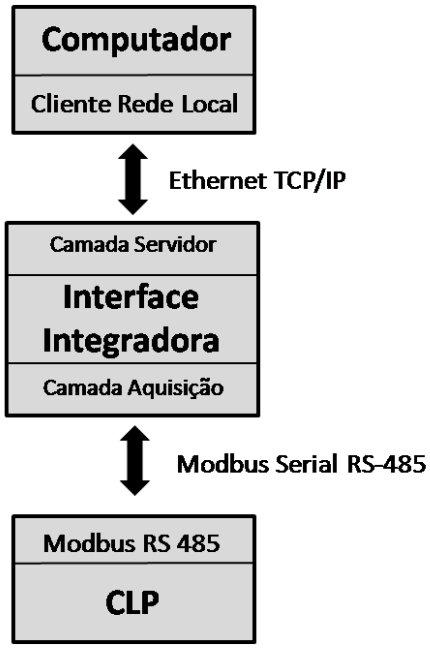
FONTE: O AUTOR (2015)

4.2 Testes em Ambiente Real

Com os resultados positivos obtidos na etapa da simulação, após foi realizado o teste do sistema completo, conectado a um CLP utilizado para automação de máquinas e processos. O CLP utilizado para o teste foi da fabricante Unitronics modelo Vision 120. Este CLP possui portas de comunicação RS-232 e RS-485 com suporte ao protocolo de comunicação Modbus. Possui também uma IHM embutida que facilita a visualização de dados da memória e a alteração destes valores, auxiliando nos testes.

Primeiramente, foram realizadas as conexões como mostra a Figura 4-6 . A interface integradora e o CLP foram conectados através da porta de comunicação RS-485 de ambos os dispositivos para realizar a aquisição dos dados, assim como a conexão do cabo de rede entre a interface integradora e o computador para acesso ao servidor e verificação dos valores adquiridos, de acordo com a Figura 4-6.

Figura 4-6 - Esquema de conexão entre os equipamentos para realizar o teste



FONTE: O AUTOR (2015)

Figura 4-7 - Ambiente real com aquisição de dados de um CLP

FONTE: UNITRONICS

A programação do CLP Vision 120 foi realizada pelo software de programação VisiLogic. Nele, foi implementado uma aplicação exemplo com 4 registradores na memória interna do CLP. O endereço desses registradores correspondem aos endereços lógicos do CLP MI101, MI102, MI103, MI104. Os valores de cada um dos registradores foram disponibilizados em uma tela da IHM, para que possam ser editados em tempo real durante o teste.

De acordo com o datasheet do equipamento, a tabela fornecida pelo fabricante informa que os endereços dos registradores de memória MI correspondem ao endereço Modbus 40001 para o endereço MI0, sendo a leitura através da função *Read Input Register* (4), devendo ser acrescentado um offset para cada endereço de memória subsequente. Assim, os endereços Modbus que devem ser lidos são 40102, 40103, 40104 e 40105.

Nesta etapa, fica clara a vantagem do software configurador da interface integradora desenvolvido. Cada fabricante especifica diferentes endereços Modbus, e a tarefa de adaptar cada aplicação as suas particularidades se tornaria muito difícil e demorado. Assim, no software configurador, foram inseridos os valores dos registradores Modbus desejados. Além disso, foi configurada a aquisição do canal digital #1 e do canal analógico #1, que correspondem ao botão "1" e ao potenciômetro, ambos localizados na placa do kit de desenvolvimento. Pode-se verificar na Figura 4-8, a tela de configuração do software configurador com os valores programados.

Após a programação, os valores dos registradores foram configurados na IHM com valores 19, 230, 166 e 50, respectivamente, como pode ser verificado na Figura 4-9. O botão '1' foi pressionado e o potenciômetro foi deslocado até o seu ponto máximo. Assim, foi realizado o acesso ao IP da interface integradora dentro do navegador da internet no computador. Os valores foram indicados corretamente pelo servidor da interface integradora, como pode-se ver na Figura 4-10.

Figura 4-8 - Configuração da Interface Integradora para Teste

FONTE: O AUTOR (2015)

Figura 4-9 - IHM do CLP indicando os valores inseridos nos registradores

FONTE: O AUTOR (2015)

Figura 4-10 - Tela do servidor visualizada no navegador, mostrando os resultados do teste

FONTE: O AUTOR (2015)

Para validar as funções do software de configuração, foi realizada a alteração dos parâmetros da interface integradora. Os endereços foram invertidos a fim de que os registradores 1, 2, 3 e 4 da interface, fossem configurados para ler os registradores R4, R3, R2, R1 do CLP, respectivamente. Na IHM, os valores foram alterados para R1=1, R2=2, R3=3 e R4=4. O potenciômetro foi deslocado até seu ponto mínimo e o sinal digital foi alterado para o botão '2', apesar de ser mantido pressionado o botão '1', devendo a entrada digital indicar o valor "OFF".

Os resultado obtidos, como pode ser visto na Figura 4-11, foram indicados corretamente pelo servidor.

Figura 4-11 - Tela do servidor, mostrando os resultados do segundo teste

FONTE: O AUTOR (2015)

5 CONCLUSÃO

A globalização, o avanço tecnológico e as demandas da indústria, exigem das empresas soluções tecnológicas que visam diminuir o atraso das respostas, melhorar a qualidade e aumentar a produtividade. Assim, este trabalho teve como objetivo desenvolver um sistema embarcado capaz de promover a integração dos dados no nível de máquina com os processos de negócios, como forma de criar novas soluções para integração da informação dentro das empresas.

Foi desenvolvido um sistema chamado de interface integradora, em um kit de desenvolvimento utilizando um microcontrolador ARM Cortex-M3, modelo LPC1768. O sistema é composto de três camadas: Camada de aquisição, camada de processamento e camada de servidor de dados. Com este sistema, tinha-se como objetivo adquirir dados de máquinas do chão de fábrica, através de protocolo de comunicação Modbus, sinais digitais e sinais analógicos, processar os dados adquiridos e criar um servidor embarcado capaz de disponibilizar os dados na rede local. Além disso, um software configurador foi desenvolvido para facilitar a configuração e personalização do equipamento para diversas situações.

Testes simulados através de software para validar cada uma das camadas da aplicação foram realizados, além de verificar o funcionamento do sistema adquirindo dados de um controlador industrial real, para que fosse possível validar o sistema em um ambiente o mais próximo de uma situação real na indústria. Conectando a interface integradora a um computador através de uma rede local, foi possível acessar e visualizar os dados em tempo real, conforme esperado. Além disso, o software de configuração foi capaz de personalizar o funcionamento da interface integradora em todos os casos. Portanto, considera-se que os resultados foram satisfatórios e os objetivos foram atingidos com sucesso.

A implementação de comunicação de dados em sistema embarcados, mostrou-se uma solução que atende o desenvolvimento de soluções dedicadas, com baixo custo em relação a computadores pessoais. Para o trabalho desenvolvido, uma série de melhorias podem ser agregadas para tornar o produto pronto para utilização e comercialização em alta escala:

- Software configurador com maior nível de personalização, como quantidade de registradores Modbus ajustável, capacidade de selecionar outras funções Modbus e outros parâmetros;

- Configurador do sistema ser integrado junto ao servidor embarcado, para que não seja necessário nenhum software adicional;

- Adicionar suporte a outros protocolos de comunicação industrial, visando aumentar as possibilidades de integração;

- Melhorar a visualização de dados com recursos avançados como: gráficos, tabelas e páginas dinâmicas;

- Implementar comunicação sem fio a rede local.

REFERÊNCIAS BIBLIOGRÁFICAS

ZURAWSKI, Richard (Ed.). **Industrial communication technology handbook.** CRC Press, 2014.

ALBUQUERQUE, Pedro Urbano Braga de; ALEXANDRIA, Auzuir Ripardo de. **Redes Industriais: aplicações em sistemas digitais de controle distribuído**. Editora Ensino Profissional. 2a edição, 2009.

KARNOUSKOS, Stamatis; COLOMBO, Armando Walter. **Architecting the next generation of service-based SCADA/DCS system of systems**. In: IECON 2011-37th Annual Conference on IEEE Industrial Electronics Society. IEEE, 2011. p. 359-364.

SAUTER, Thilo. **The continuing evolution of integration in manufacturing automation**. Industrial Electronics Magazine, IEEE, v. 1, n. 1, p. 10-19, 2007.

TEXAS INSTRUMENTS. **32-Bit ARM® Cortex®-M4F MCU-Based Small Form Factor Serial-to-Ethernet Converter**, 2014.

COPELAND, Karen AF. **Statistical Process Control in Manufacturing Practice**. Journal of Quality Technology, v. 30, n. 3, p. 305, 1998.

BROOKS, Paul. **Ethernet/IP-industrial protocol**. In: Emerging Technologies and Factory Automation, 2001. Proceedings. 2001 8th IEEE International Conference on. IEEE, 2001. p. 505-514.

LUGLI, A.B.,**Uma visão do protocolo industrial Profinet e suas aplicações**, http://**www**.inatel.br, acessado em 27/09/2015.

SAP, **Transform, and Reimagine: SAP® Solutions for the Internet of Things**, 2015

WORLD ECONOMIC FORUM, **Industrial Internet of Things: Unleashing the Potential of Connected Products and Services**, 2015

SPIESS P.; KARNOUSKOS, S., **Maximizing the business value of networked embedded systems through process-level integration into enterprise software.** InProc. 2nd International Conference on Pervasive Computing and Applications, 2007.

STALLINGS, William. **Arquitetura e organização de computadores: projeto para o desempenho**. Prentice-Hall, 2006.

SPECIFICATION, Modbus Application Protocol. V1. 1b3. **Hopkinton: Modbus Organization, Inc., Abril**, v. 26, 2012.

TANENBAUM, Andrew S. **Redes de Computadores**, 1999. Editora Campus, 4ª edição.

AUTOMATION, S. MODBUS over serial line–Specification and Implementation guide. V, 2002.

METCALFE, Robert M.; BOGGS, David R. Ethernet: distributed packet switching for local computer networks. **Communications of the ACM**, v. 19, n. 7, p. 395-404, 1976.

NXP, B. V. UM10360–LPC17xx User Manual. 2010.

GODSE, DA Godse AP. **Microprocessors & Microcontrollers**. Technical Publications, 2008.

YUE, Yang; ZHANG, C. G.; YUANG, A. J**. Design and implementation of embedded man-machine interface based on Modbus**. Industrial Control Computer, v. 19, n. 1, p. 8-10, 2006.

LEE, Kyung Chang; LEE, Suk. Performance evaluation of switched Ethernet for real-time industrial communications**. Computer standards & interfaces**, v. 24, n. 5, p. 411-423, 2002.

GUI, Zhang; GUOQIANG, Jin; HUI, Li. **Research and Implementation of Modbus RTU Protocol Based on ARM** [J]. Electric Power Science and Engineering, v. 1, p. 008, 2011.

APÊNDICE A – Código Camada de Aquisição

A.1 – Arquivo Modbus.h

```
typedef struct {
	unsigned char address;
	unsigned char cmd;
	unsigned char data[252];
	unsigned short crc;
	unsigned char dataSize;
} MbRTU;



void Mb_Init(int verbose);
void Mb_ReadInputRegisters(uint8_t slaveAddress, uint16_t startAddress, uint16_t quantity, char address, int verbose);
void Mb_ReceiveMsg(void);
void Mb_IncrementTimer(void);
void Mb_SendMessage(MbRTU *ptr);
void ImprimeVetorSerial(const char *vetor, int length);
void LCDPrintFrame(MbRTU *vetor, int length);
void ImprimeDisplay(const char *vetor, int length);
void getModbusData(int *MbData);
```

A.2 – Arquivo Modbus.c

```
#include "lpc17xx.h"
#include "monitor.h"
#include "timer.h"
#include "type.h"
#include "ModBus.h"
#include "uart.h"
#include "crc.h"
#include "lcd.h"

static enum
{
    INICIANDO,
	AGUARDANDO,
    RECEPCAO,
    VERIFICACAO,
	PROCESSAMENTO,
	TRANSMISSAO
} MbState = INICIANDO;

unsigned int Timer35TC=0;
```

```
extern volatile uint32_t UART3Count;
extern volatile uint8_t UART0Buffer[BUFSIZE], UART1Buffer[BUFSIZE],
UART3Buffer[BUFSIZE];
MbRTU mbFrame;
int dados[252];

char MbFlag_35TC;

//================================================================================
//------------------------------------ Public Functions ------------------------
//================================================================================

void Mb_Init(int verbose)
{
	MbState = INICIANDO;
	Timer35TC = 50;

	while (MbFlag_35TC == 0);
	MbState = AGUARDANDO;
	MbFlag_35TC = 0;

	if (verbose)
		xprintf("\nSistema Iniciado...");
}

void Mb_ReadInputRegisters(uint8_t slaveAddress, uint16_t startAddress, uint16_t
quantity, char address, int verbose)
{
	MbRTU str;
	char i;

	MbState = TRANSMISSAO;

	if (verbose)

	str.address = slaveAddress;
	str.cmd = 0x03;
	str.data[0] = (uint8_t)(startAddress>>8);
	str.data[1] = (uint8_t)(startAddress&0x00FF);
	str.data[2] = (uint8_t)(quantity>>8);
	str.data[3] = (uint8_t)(quantity&0x00FF);
	str.dataSize = quantity*2;

	Mb_SendMessage(&str);
	if (verbose)
	Timer35TC = 7;

	while (MbFlag_35TC == 0);
	MbFlag_35TC = 0;
		if (verbose)
		xprintf("\nModbus Send OK...");

	MbState = AGUARDANDO;

	while (MbFlag_35TC == 0);
	MbFlag_35TC = 0;

	if (verbose)
		xprintf("\nModbus Recepcao OK...");
```

```
	Mb_ReceiveMsg();

	MbState = AGUARDANDO;

	dados[address] = ((mbFrame.data[0]<<8) | mbFrame.data[1]);
	xprintf("\nRegistrador[%d] ... %d", address, dados[address]);

}


void Mb_ReceiveMsg(void){

	unsigned char i;

	if (UART3Count > 1){

			for(i=0; i<UART3Count; i++)
			{
				xprintf("\n%d",UART3Buffer[i]);
			}
			mbFrame.address = UART3Buffer[0];
			mbFrame.cmd = UART3Buffer[1];
			mbFrame.dataSize = UART3Buffer[2];
			for(i=3; i<(UART3Buffer[2]); i++)
			{
				mbFrame.data[i-3] = UART3Buffer[i];
			}
			mbFrame.crc = ((UART3Buffer[UART3Count-1]<<8) |

	UART3Buffer[UART3Count-2]);
	UART3Count=0;
	}
}

char Mb_CheckFrame(void){
	uint16_t crc;
	char erro=1;

	crc = crc_getCRCModBusRTU((uint8_t *)mbFrame, 6);

	if (mbFrame.crc == crc)
		erro = 0;

	return erro;

}



void Mb_SendMessage(MbRTU *ptr)
{
	uint8_t i;
	uint16_t crc;
	char BufferTX[256];


	BufferTX[0] = ptr->address;
```

```
	BufferTX[1] = ptr->cmd;

	xprintf("\n%d\n%d",BufferTX[0],BufferTX[1]);
	for(i=0; i<4; i++)
	{
		BufferTX[i+2] = ptr->data[i];
	}
	crc = crc_getCRCModBusRTU((uint8_t *)BufferTX, 6);

	BufferTX[6] = (char)(crc&0x00FF);
	BufferTX[7] = (char)(crc>>8);

	//LPC_UART3->IER = IER_THRE | IER_RLS;
	UARTSend( 3, (uint8_t *)BufferTX, 8);
	//LPC_UART3->IER = IER_THRE | IER_RLS | IER_RBR;
}


void Mb_CaracterReceived(void)
{
		if (MbState == AGUARDANDO | MbState == RECEPCAO){
			Timer35TC = 12;
			MbState = RECEPCAO;
		}else
			UART3Count=0;
}

void Mb_IncrementTimer(void)
{
	if(Timer35TC)
	{
		Timer35TC--;
		if(Timer35TC==0)
		{
			MbFlag_35TC=1;
		}
	}
}


void getModbusData(int *MbData) {

	  char i;
	  for (i=0; i<4 ; i++)
		MbData[i] = dados[i];

}
```

APÊNDICE B – Código Camada de Processamento

B.1 – Arquivo main.c

```
#include "lpc17xx.h"
#include "AppLcd.h"
#include "Beep.h"
#include "monitor.h"
#include "system_LPC17xx.h"
#include "timer.h"
#include "uart.h"
#include "usb.h"
#include "Modbus.h"
#include "inputs.h"

extern void TCPClockHandler(void);

extern unsigned int startAddress[4];

volatile DWORD TimeTick  = 0;


/* SysTick interrupt happens every 10 ms */
void SysTick_Handler (void) {
  TimeTick++;
  if (TimeTick >= 100) {
    TimeTick = 0;
    LPC_GPIO2->FIOPIN ^= 1 << 0;
    TCPClockHandler();
  }
}


void SystemInitialize(void)
{
      SystemInit();
      SysTick_Config(SystemFrequency/1000 - 1); /* Generate interrupt each 1 ms
*/
      UARTInit(0, 19200);
      UARTInit(3, 19200);
      AppLcd_Init();
      Inputs_Init();
      Timer_Init();
      Mb_Init(1);
      TCPInit();
}

void SystemConfig(void)
{
            char i=0;
            char buffer=0;
            LCD_ShowString(40, 50, "Modo Configuracao");

            while(!Inputs_GetState(1)){
            ConfigHandler();
            UART0Count++;
            }
```

```
}

int main (void)
{
	unsigned char MainSlot=0;
	SystemInitialize();

	if (Inputs_GetState(0))
	SystemConfig();

	while(1)
	{

		if(Timer_VerifyTimer(TIMER_5MS, BASE_5MS))
		{
			Timer_ResetTimer(TIMER_5MS);
			MainSlot++;
			switch(MainSlot)
			{
			case 1:
				IOHandler();
				break;

			case 2:

				ServerHandler();
				break;

			case 3:
				Mb_ReadInputRegisters(1,startAddress1,address,1);
				break;

			default:
				MainSlot = 0;
				break;
			}
		}
	}
}
```

B.3 – Arquivo Config.h

```
const unsigned char HTTPHeader[] =
{
  "HTTP/1.0 200 OK\r\n"
"Content-Type: text/html\r\n"
"\r\n"
};

const unsigned char IdLabVIEW[] =
{
  "13ConfigActived"
};

void ConfigHandler(void);
void ConfigServer(void);
```

```
unsigned char *PWebSide;
unsigned int HTTPBytesToSend;
```

B.3 – Arquivo Config.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "lpc17xx.h"

#define extern
#include "server.h"
#include "modbus.h"
#include "lcd.h"
#include "type.h"
#include "EMAC.h"
#include "tcpip.h"

#include "webpage.h"

unsigned int pagecounter = 100;
unsigned int adcValue    =   0;
unsigned int lcdX    =  40;


unsigned int startAddress[4] = {5,6,7,8};

void ConfigHandler(void){

      xprintf("\nConfig....");

    if (!(SocketStatus & SOCK_ACTIVE)) TCPPassiveOpen();
    EthernetHandler();
    ConfigServer();

}

void ConfigServer(void)
{

unsigned char *Key;
             char NewKey[5];
             char i;
             char *teste;
             char ptr[4];

unsigned char RxBuffer[MAX_TCP_TX_DATA_SIZE];
char statusConfig=0;

Key = TCP_TX_BUF;

  if (SocketStatus & SOCK_CONNECTED){
    if (SocketStatus & SOCK_DATA_AVAILABLE){
             memcpy(RxBuffer, TCP_RX_BUF, TCPRxDataCount);
             TCPReleaseRxBuffer();

             if (RxBuffer[0]=='I'){
```

```
            if (RxBuffer[1]=='I')
                  if (RxBuffer[2]=='0'){
                  xprintf("\n\n\nINICIO OK\n\n\n.");
                  if (SocketStatus & SOCK_TX_BUF_RELEASED){
                  memcpy(TCP_TX_BUF,IdLabVIEW,sizeof(IdLabVIEW) - 1);
                  TCPTxDataCount = sizeof(IdLabVIEW) -1;
                  TCPTransmitTxBuffer();
                        }
                  }

      }else if (RxBuffer[0]=='%'){
                   if (RxBuffer[1]=='R')
                        switch(RxBuffer[2]){

                         case '0' :{
                               xprintf("\n\n\nADDRESS\n\n\n.");
                               TCPTxDataCount = 3;
                               sprintf(NewKey, "%02u", 19);
                               memcpy(Key, NewKey, 2);
                               Key+=2;

                               for (i=0;i<4;i++){
                                 sprintf(NewKey,"%04u",startAddress[i]);
                                  memcpy(Key+(i*4), NewKey, 4);
                                 TCPTxDataCount += 4;
                               }
                                TCPTransmitTxBuffer();
                              break;
                        }

                  }
      }
```

APÊNDICE C – Código da Camada de Servidor de Dados

C.1 – Arquivo Server.h

```
const unsigned char HTTPHeader[] =
{
  "HTTP/1.0 200 OK\r\n"
"Content-Type: text/html\r\n
"\r\n
};

const unsigned char IdLabVIEW[] =
{
  "13ConfigActived"
};
```

```
void TCPInit(void);
void ServerHandler(void);
void InitOsc(void);
void InitPorts(void);
void HTTPServer(void);
void InsertDynamicValues(void);
unsigned int GetAD7Val(void);


unsigned char *PWebSide;
unsigned int HTTPBytesToSend;
unsigned char HTTPStatus;
#define HTTP_SEND_PAGE              0x01
```

C.2 – Arquivo Server.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "lpc17xx.h"

#define extern

#include "server.h"
#include "modbus.h"
#include "lcd.h"
#include "type.h"
#include "EMAC.h"
#include "tcpip.h"

#include "webpage.h"

unsigned int pagecounter = 100;
unsigned int adcValue    =   0;
unsigned int lcdX    =  40;


unsigned int startAddress[4] = {5,6,7,8};


void TCPInit(void){

  LPC_GPIO0->FIODIR   |= 1 << 21;
  LPC_GPIO0->FIOPIN   |= 1 << 21;

  LPC_GPIO2->FIODIR   |= 1 << 0
  LPC_PINCON->PINSEL3 |=  (3ul<<30);
  LPC_SC->PCONP       |=  (1<<12);
```

```
  LPC_ADC->ADCR        =  (1<< 5) | (4<< 8) |(1<<21);

  TCPLowLevelInit();

  HTTPStatus = 0
  TCPLocalPort = TCP_PORT_HTTP
 }


void ServerHandler(void){

    if (!(SocketStatus & SOCK_ACTIVE)) TCPPassiveOpen();
    EthernetHandler();
    HTTPServer();

}


void HTTPServer(void)
{
  if (SocketStatus & SOCK_CONNECTED)              {
    if (SocketStatus & SOCK_DATA_AVAILABLE)
        TCPReleaseRxBuffer();
    if (SocketStatus & SOCK_TX_BUF_RELEASED)
      if (!(HTTPStatus & HTTP_SEND_PAGE))
      {
        HTTPBytesToSend = sizeof(WebSide) - 1
        PWebSide = (unsigned char *)WebSide;
}

      if (HTTPBytesToSend > MAX_TCP_TX_DATA_SIZE)
      {
        if (!(HTTPStatus & HTTP_SEND_PAGE)){
          memcpy(TCP_TX_BUF, HTTPHeader, sizeof(HTTPHeader) - 1);
          memcpy(TCP_TX_BUF+sizeof(HTTPHeader)-1, PWebSide, MAX_TCP_TX_DATA_SIZE -
sizeof(HTTPHeader) + 1);
          HTTPBytesToSend -= MAX_TCP_TX_DATA_SIZE - sizeof(HTTPHeader) + 1;
          PWebSide += MAX_TCP_TX_DATA_SIZE - sizeof(HTTPHeader) + 1;
        }
        else
        {
          memcpy(TCP_TX_BUF, PWebSide, MAX_TCP_TX_DATA_SIZE);
          HTTPBytesToSend -= MAX_TCP_TX_DATA_SIZE;
          PWebSide += MAX_TCP_TX_DATA_SIZE;
        }

        TCPTxDataCount = MAX_TCP_TX_DATA_SIZE;   // bytes to xfer
        InsertDynamicValues();                   // exchange some strings...
        TCPTransmitTxBuffer();                   // xfer buffer
      }
      else if (HTTPBytesToSend)                  // transmit leftover bytes
      {
        memcpy(TCP_TX_BUF, PWebSide, HTTPBytesToSend);
        TCPTxDataCount = HTTPBytesToSend;        // bytes to xfer
        InsertDynamicValues();                   // exchange some strings...
        TCPTransmitTxBuffer();                   // send last segment
        TCPClose();                              // and close connection
        HTTPBytesToSend = 0;                     // all data sent
      }
```

```
      HTTPStatus |= HTTP_SEND_PAGE;
    }
  }
  else
    HTTPStatus &= ~HTTP_SEND_PAGE;
  }


unsigned int GetAD7Val(void)
{
  unsigned int val;
  LPC_ADC->ADCR |=  (1<<24);
  while (!(LPC_ADC->ADGDR & (1UL<<31)));
  val = ((LPC_ADC->ADGDR >> 4) & 0xFFF);
  LPC_ADC->ADCR &= ~(7<<24);
  return(val);
}


void InsertDynamicValues(void)
{
  unsigned char *Key;
           char NewKey[5];
  unsigned int i;
  int MbData[4];
  getModbusData(&MbData[0]);

  if (TCPTxDataCount < 4) return
  Key = TCP_TX_BUF;


  for (i = 0; i < (TCPTxDataCount - 3); i++)
  {
    if (*Key == 'A'){
     if (*(Key + 1) == 'D')
       if (*(Key + 3) == '%')
         switch (*(Key + 2))
         {
           case '8' :
           {
             adcValue = GetAD7Val();
             sprintf(NewKey, "%5u", adcValue);
             memcpy(Key, NewKey, 5);
             break;
           }
           case '0' :
           {
             sprintf(NewKey, "%3u", (adcValue*100)/4024);
             memcpy(Key, NewKey, 3);
             break;
           }
                case '1' :
           {

             sprintf(NewKey, "%3u", ++pagecounter);
             memcpy(Key, NewKey, 3);
                     *(Key + 3) = ' ';
             break;
```

```
          }
         }
        if (*(Key + 1) == 'M')
         if (*(Key + 3) == '%')
          switch (*(Key + 2))
          {
           case '0' :
            {
             sprintf(NewKey, "%5u", MbData[0]);
             memcpy(Key, NewKey, 5);
             break;
            }

             case '1' :           {
             sprintf(NewKey, "%5u", MbData[1]);
             memcpy(Key, NewKey, 5);
             break;
            }

             case '2' :
            {
             sprintf(NewKey, "%5u", MbData[2
             memcpy(Key, NewKey, 5);
             break;
            }

             case '3' :
            {
             sprintf(NewKey, "%5u", MbData[3]);
             memcpy(Key, NewKey, 5);
             break;
            }
          }
       }
    Key++;
  }
}
```