UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
CAMPUS DE ILHA SOLTEIRA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

IMPLEMENTAÇÃO DE UM NÓ IEEE 1451, BASEADO EM FERRAMENTAS ABERTAS E PADRONIZADAS, PARA APLICAÇÕES EM AMBIENTES DE INSTRUMENTACÃO DISTRIBUÍDA

Silvano Renato Rossi

Tese de Doutorado submetida à Universidade Estadual Paulista - UNESP, Campus de Ilha Solteira, como parte dos requisitos necessários para a obtenção do título de Doutor em Engenharia Elétrica.

Orientador: Prof. Dr. Aparecido Augusto de Carvalho

Co-Orientador: Prof. Dr. Alexandre César Rodrigues da Silva

ILHA SOLTEIRA – SP, JANEIRO DE 2005

Implementação de um Nó IEEE 1451, Baseado em Ferramentas Abertas e Padronizadas, para Aplicações em Ambientes de Instrumentação Distribuída

SILVANO RENATO ROSSI

TESE SUBMETIDA À FACULDADE DE ENGENHARIA - CAMPUS DE ILHA SOLTEIRA – UNESP – COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO TÍTULO DE DOUTOR EM ENGENHARIA ELÉTRICA

COMISSÃO EXAMINADORA:

Prof. Dr. Marcelo Carvalho Minhoto Teixeira
Vice-Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Prof. Dr. Aparecido Augusto de Carvalho – orientador

Prof. Dr. Onofre Trindade Júnior

Prof. Dr. Edward David Moreno Ordonez

Prof. Dr. Cláudio Kitano

Prof. Dr. Ricardo Tokio Higuti

Ilha Solteira-SP, janeiro de 2005.

A Deus

A minha mãe, Rosa

À memória do meu pai, Roberto

Aos meus irmãos, Adriana e Claudio

Ao meu amor, Diva

Ao colega e amigo, Edson Antonio Batista

DEDICO

AGRADECIMENTOS

A Deus, pelo amor, pela misericórdia, pelas forças tanto nos momentos bons como nos difíceis.

À minha família, pelo apoio incondicional ao longo da realização deste trabalho.

Ao meu orientador, Professor Aparecido Augusto de Carvalho, pela orientação, pela motivação, pela confiança, pela sinceridade, pela humildade, pela amizade e, principalmente, por contribuir para o meu crescimento, não apenas como profissional, mas também como pessoa.

Ao meu co-orientador, Professor Alexandre César Rodrigues da Silva, pela orientação, pelas sugestões e contribuições, pela confiança e pelos bons momentos compartilhados.

Ao Professor Cláudio Kitano, por ser um modelo de profissional, que contribuiu muito no desenvolvimento da tese e que incentivou, em mim, o espírito de pesquisador.

Ao engenheiro eletricista Edson Antonio Batista, colega e amigo, porque sem a sua ajuda este trabalho não teria sido realidade.

Aos Professores Marcelo Carvalho Minhoto Teixeira, Ricardo Tokio Higuti, José P. Fernandes Garcia, Edward D. Moreno Ordonez e Onofre Trindade Júnior, pelas sugestões e contribuições.

Ao Programa de Pós Graduação em Engenharia Elétrica da FEIS-UNESP, pela oportunidade; aos professores do Programa, pela qualidade de ensino e aos servidores técnico-administrativos, pela atenção e disposição.

À "Universidad Nacional del Centro de la Pcia. de Buenos Aires", pelo apoio financeiro cedido através da bolsa do "Programa VII", para aperfeiçoamento em docência e pesquisa.

Aos Professores Marcelo Spina, Roberto Juan de La Vega, Gerardo Acosta e ao grupo Intelymec da "Universidad Nacional del Centro de la Pcia. de Buenos Aires", pelo apoio e confiança.

À Fundunesp, pelo apoio financeiro cedido para compra de equipamentos e materiais.

À minha namorada Diva e sua família, pelo amor, paciência e ajuda.

Aos colegas, Uender da Costa Faria, Tony Inácio da Silva, Josivaldo Godoy da Silva, Adriano dos Santos Cardoso e ao pessoal do Laboratório de Processamento de Sinais e Sistemas Digitais, pelas sugestões e pela disposição.

“Graças vos dou, porque me ouvistes,
e vos fizestes meu salvador”
(Sal. 117, 21)

RESUMO

Atualmente, as redes de transdutores inteligentes desempenham um papel de importância vital em sistemas de Medição e Controle Distribuído. Nesse contexto, o Padrão IEEE 1451 para interfaceamento de transdutores inteligentes tem como objetivo simplificar a conectividade de transdutores em ambientes de rede, fornecendo, para tal fim, um conjunto de interfaces padronizadas, aumentando a flexibilidade dos sistemas de instrumentação distribuída. Neste trabalho descreve-se a implementação de um nó de rede em conformidade com o padrão IEEE 1451. O nó foi completamente desenvolvido através do emprego de ferramentas padronizadas e sistemas abertos. O nó é composto por um Processador de Aplicação com Capacidade de Operar em Rede (NCAP), com base no padrão IEEE 1451.1 e um Módulo de Interface para Transdutores Inteligentes (STIM), em conformidade com o padrão IEEE 1451.2. A parte física do NCAP foi implementada através dos recursos de um Computador Pessoal (PC) e de um Dispositivo Lógico Programável (PLD) de uso geral. A parte lógica do NCAP foi desenvolvida através da tecnologia Java. O STIM foi implementado com dispositivos lógicos programáveis versáteis, de uso geral, e sua funcionalidade foi integralmente descrita em linguagem de descrição de *hardware*. O conjunto NCAP-STIM foi conectado a uma rede de área local, sob o modelo de comunicação cliente-servidor, sendo que várias aplicações clientes podem acessar as informações dos transdutores conectados ao STIM, através da rede, via intermediação do NCAP. O emprego de ferramentas padronizadas e abertas no desenvolvimento total do sistema IEEE 1451 é uma das contribuições mais importantes do presente trabalho. No entanto, há várias contribuições pontuais como: a maneira de descrever as Informações de Transdutores em Formato Eletrônico (TEDS), a implementação de um gerenciador de protocolo que permite o uso da porta paralela de um PC sem qualquer modificação da mesma, a utilização de PLDs de baixo custo para a implementação do STIM e do gerenciador de protocolo, e o desenvolvimento do *software* do NCAP integralmente na linguagem Java. Através da implementação do padrão IEEE 1451, as indústrias podem obter inúmeros benefícios como redução de custos de sistemas de instrumentação devido à redução da complexidade, flexibilidade para transmitir informações através de uma rede de comunicação e maior facilidade para ampliar os sistemas sem efetuar configurações complicadas.

Palavras chave **-** Medição e Controle Distribuído, Transdutores em rede, Interfaceamento, IEEE 1451, STIM, NCAP, Transdutor Inteligente, Tecnologia de Lógica Programável, Java.

ABSTRACT

Nowadays, smart transducer networks play an essential role in distributed measurement and control systems. In this context, the IEEE 1451 smart transducer interface standards aimed to simplify transducer connectivity, providing a set of common interfaces for connecting transducers in a networked fashion, increasing the flexibility of distributed instrumentation systems. In this work the implementation of a network node according to the IEEE 1451 standard is introduced. The node has been fully developed using open and standardized tools. A Network Capable Application Processor (NCAP) according to the IEEE 1451.1 Standard and a Smart Transducer Interface Module (STIM) comprises the node. The physical part of the NCAP has been implemented using the resources of a Personal Computer (PC) and a general-purpose Programmable Logic Device (PLD). The logical part of the NCAP has been developed using Java technology. The STIM module was implemented with versatile, general-purpose Programmable Logic Devices. STIM functionality has been fully developed in hardware description language. A network node (STIM-NCAP) was connected in a client-server model-based local area network. Many client applications can access STIM transducers information, through the network with the NCAP as an intermediary. One of the most important contributions of this work is the employment of open and standardized tools for implementing the IEEE 1451 network node. However, there are many specific contributions such as: Transducer Electronic Data Sheet (TEDS's) description method, programmable logic-based Protocol Manager implementation that allows the use of the parallel port without any modification, the employment of low-cost PLDs for implementing the STIM and the Protocol Manager, and Java-based NCAP software development. Through the implementation of the IEEE Standard, industries can obtain great benefits such as optimization of the instrumentation system costs due to reduced complexity, flexibility for transmitting information through a communication network, and more flexibility for enhancing their systems without complicated configuration tasks.

Keywords - Distributed Measurement and Control, Transducer Networks, Interfacing, IEEE 1451, STIM, NCAP, Smart Transducer, Programmable Logic Technology, Java.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADC (A/D)	*Analog to Digital Converter* (Conversor Analógico-Digital)
API	*Application Programming Interfaces* (Interfaces de Programação de Aplicativos)
ASIC	*Application-Specific Integrated Circuit* (Circuito Integrado de Aplicação Específica)
BIOS	*Basic Input Output System* (Sistema Básico de Entrada Saída)
CMOS	*Complementary Metal-Oxide Semiconductor* (Semicondutor Metal-Óxido Complementar)
COMM	*Communication* (Comunicação)
CPLD	*Complex Programmable Logic Device* (Dispositivo Lógico Programável Complexo)
CRC	*Cyclical Redundancy Check* (Teste de Redundância Cíclica)
CSMA/CD	*Carrier Sense Multiple Access with Collision Detection* (Acesso Múltiplo com Detecção de Portadora, com Detecção de Colisão)
DAC(D/A)	*Digital to Analog Converter* (Conversor Digital-Analógico)
ECP	*Extended Capability Port* (Porta com Capacidade Estendida)
EEPROM	*Electrically-Erasable Programmable Read-Only Memory* (Memória Exclusivamente de Leitura, Eletricamente Apagável)
EPP	*Enhanced Parallel Port* (Porta Paralela Estendida)
EPROM	*Erasable Programmable Read-Only Memory* (Mémoria Exclusivamente de Leitura, Programável e Apagável)
FPGA	*Field-Programmable Gate Array* (Matriz de Portas Programável em Campo)
GAL	*Generic Array Logic* (Matriz Lógica Genérica)
HDL	*Hardware Description Language* (Linguagem de Descrição de Hardware)
IC	*Integrated Circuit* (Circuito Integrado)
IEC	*International Electrotechnical Commission* (Comissão Eletrotécnica Internacional)
I/O (E/S)	*Input/Output* (Entrada/Saída)
IR	*InfraRed* (Infravermelho)
ISO	*International Standards Organization* (Organização Internacional de Padronização)
ISP	*In System Programmability* (Programação no Sitema)
ISS	*Integrated Smart Sensor* (Sensor Inteligente Integrado)

JDK	*Java Development Kit* (Kit de Desenvolvimento Java)
JNI	*Java Native Interface* (Interface Java Nativa)
JRE	*Java Run-Time Environment* (Ambiente para Execução Java)
JTAG	*Joint Test Action Group*
JVM	*Java Virtual Machine* (Máquina Virtual Java)
LAN	*Local Area Network* (Rede de Área Local)
LLC	*Logical Link Control* (Controle de Enlace Lógico)
LPM	*Library of Parameterized Modules* (Biblioteca de Módulos Parametrizáveis)
MAN	*Metropolitan Area Network* (Rede de Área Metropolitana)
MEMS	*Micro-Electro-Mechanical System* (Sistema Microeletromecânico)
MPS	*Michigan Parallel Standard* (Padrão Michigan Paralelo)
MPU	*Microprocessor Unit* (Unidade Microprocessadora)
MSS	*Michigan Serial Standard* (Padrão Michigan Série)
NCAP	*Network Capable Application Processor* (Processador de Aplicação com Capacidade de Operar em Rede
NIST	*National Institute of Standards and Technology* (Instituto Americano de Padrões e Tecnologia)
OSI	*Open Systems Interconnection (Interconexão de Sistemas Abertos)*
PAL	*Programmable Array Logic* (Matriz Lógica Programável)
PC	*Personal Computer* (Computador Pessoal)
PLC	*Programmable Logic Controller* (Controlador Lógico Programável)
PLD	*Programmable Logic Device* (Dispositivo Lógico Programável)
RMI	*Remote Method Invocation* (Chamada de Método Remoto)
RPC	*Remote Procedure Calls* (Chamada de Procedimentos Remotos)
SPI	*Serial Peripheral Interface* (Interface Série para Periféricos)
SPP	*Standard Parallel Port* (Porta Paralela Padrão)
SRAM	*Static Random Access Memory* (Memória Estática de Acesso Aleatório)
STIM	*Smart Tranducer Interface Modue* (Módulo de Interface para Transdutores Inteligentes)
TC-9	*Technical Committee 9 on Sensor Technology* (Comitê Técnico 9 em Tecnologia de Sensores)
TCP/IP	*Transmission Control Protocol/Internet Protocol* (Protocolo de Controle de Transmissão/Protocolo de Internet)
TEDS	*Transducer Electronic Data Sheet* (Especificações de Transdutor em em Formato Eletrônico)
TII	*Transducer Independent Interface* (Interface Independente de Transdutores)
TTL	*Transistor-Transistor Logic* (Lógica Transistor-Transistor)
VHDL	*Very High Speed Integrated Circuit - Hardware Description Language* Linguagem de Descrição de Hardwrare para Circuitos Integrados de Muito Elevada Velocidade)

SUMÁRIO

Capítulo 1

Introdução Geral

1.1 - Introdução

Os dispositivos sensores e os dispositivos atuadores são os protagonistas no cenário da automação industrial. O fato de vivermos atualmente em um mundo de informações e comunicações, exige não apenas que as informações relacionadas com os transdutores em um determinado ambiente sejam transmitidas de um ponto a outro, mas também compartilhadas. Nasce, assim, o conceito de rede de transdutores.

Impulsionados pelo avanço significativo das tecnologias de redes de computadores, o crescimento acentuado da Internet e a grande disponibilidade de ferramentas para desenvolvimento de sistemas digitais, os sistemas de medição e controle distribuído, constituem hoje, uma tendência, não apenas no âmbito industrial, mas também na indústria aeroespacial, automação residencial e engenharia biomédica.

Os transdutores em rede fazem parte desses sistemas e a transmissão de suas informações, de maneira adequada, é um fato relevante. Os sistemas de medição e controle distribuídos precisam ser flexíveis, visando a integração dos componentes do sistema. Isto significa que, desde os transdutores até o receptor final das informações deve-se estabelecer uma interação cada vez mais simples e acessível. Neste sentido, o projeto das interfaces envolvidas no sistema desempenha um papel de importância vital. O problema do interfaceamento demandou, e demanda ainda, grandes esforços por parte dos engenheiros projetistas da área de instrumentação.

Na década de 1980, a evolução das comunicações digitais possibilitou a introdução dos denominados barramentos de campo na área de automação industrial. Esses sistemas de interconexão possibilitam a comunicação digital bidirecional sob um determinado protocolo de comunicação, permitindo que os dispositivos transdutores conectados ao barramento possam trocar informações. Entretanto, existem atualmente inúmeras tecnologias disponíveis, muitas

delas proprietárias [1]. Este fato possibilitou um notável avanço na automação industrial, porém, acarretou um outro problema difícil de se resolver, o interfaceamento entre os transdutores e as diferentes tecnologias de rede.

Reconhecendo-se a necessidade destes problemas serem resolvidos, no final da década de 1990 foram aprovadas as primeiras diretrizes do padrão de interfaceamento para transdutores inteligentes IEEE 1451 [2], [3]. Esta norma introduz um conceito inovador para estabelecer a independência entre um nó de rede envolvendo transdutores e o protocolo de comunicação da rede. Através dessas normas especificam-se as bases para implementar um modelo de informação de um transdutor inteligente, baseando-se na programação orientada a objetos, que hoje, junto à utilização de padrões, sistemas abertos de comunicação e ferramentas associadas à Internet, constituem as novas tendências na área de instrumentação distribuída [4].

A exploração de ferramentas e recursos de domínio aberto e padronizados, e sua aplicação em sistemas de instrumentação são as principais idéias que norteiam os objetivos deste trabalho.

1.2 - Breve Histórico do Padrão IEEE 1451

Foi em 1993, quando o Comitê Técnico em Tecnologia de Sensores (TC-9 - *Technical Committee on Sensor Technology*) da Sociedade de Instrumentação e Medidas do IEEE (IEEE *Instrumentation and Measurement Society*) começou a trabalhar na definição de um padrão de interfaceamento para conexão de transdutores inteligentes em rede.

Em parceria com o Instituto Americano de Padrões e Tecnologia (NIST - *National Institute of Standards and Technology*), o IEEE iniciou uma série de *workshops* sobre padronização de interfaces, que deram origem aos grupos de trabalho que iriam criar as especificações pertencentes à família 1451 [5].

No mês de março de 1994 foi realizado o primeiro *workshop* do NIST/IEEE sobre interfaces para sensores inteligentes. Já no mês de setembro do mesmo ano teve lugar o segundo w*orkshop*, onde foi demonstrado o conceito de especificações de transdutores em formato eletrônico. Esses eventos motivaram a formação de dois grupos de trabalho denominados IEEE P1451.1 e IEEE P1451.2[1] respectivamente. No terceiro *workshop*, realizado no mês de maio de 1995 foram

[1] A letra P significa em fase de projeto.

demonstrados os primeiros conceitos de interoperabilidade e *plug and play*[2] em redes de transdutores. No mês de setembro de 1996 foram criados os grupos de trabalho IEEE P1451.3 e IEEE P1451.4 .

No ano de 1997 foi aprovado o padrão IEEE Std. 1451.2 (*Transducer to Microprocessor Communication Protocols and Transducers Electronic Data sheet (TEDS) Formats*) [2]. Esta norma foi criada com o objetivo de padronizar a interface entre transdutores e processadores de rede, definindo, para tal fim, o Módulo de Interface para Transdutores Inteligentes (STIM - *Smart Transducer Interface Module*), a Interface Independente de Transdutores (TII - *Transducer Independent Interface*) e a maneira de serem consideradas as especificações de transdutores para serem gravadas em formato eletrônico (TEDS – *Transducer Electronic Data Sheet*). O STIM deve conter um processador para implementar as funcionalidades descritas neste padrão, uma memória de programa para armazenar as estruturas TEDS, circuitos de condicionamento e conversão de sinal e os transdutores propriamente ditos.

No ano de 1999 foi aprovado o padrão IEEE Std. 1451.1 (*Network Capable Application Processor (NCAP) Information Model for Smart Transducers*) [3]. O padrão IEEE 1451.1 foi criado com o objetivo de padronizar o *software* da interface entre processadores de aplicação para operar em rede (NCAPs) e redes de controle. Isto possibilitou a independência entre um nó de rede e o protocolo de comunicação da rede. A padronização é atingida através do desenvolvimento de um conjunto comum de informações ou modelo objeto, incluindo: blocos de transdutores, blocos de funções e blocos de rede. Este modelo de informação é baseado na programação orientada a objetos.

No ano de 2003 foi aprovada a diretriz IEEE Std. 1451.3 (*Digital Communication and Transducer Electronic Data Sheet (TEDS) Formats for Distributed Multidrop Systems*) [6]. Nesta norma é especificada a forma de comunicação para um arranjo de vários transdutores distribuídos, cujas informações precisam ser lidas de forma sincronizada através de um barramento conectado ao NCAP.

No ano de 2004 foi aprovado o padrão IEEE Std. 1451.4 (*Mixed-mode Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats*) [7]. Nesta norma especifica-se como um sinal analógico associado a um transdutor e um sinal digital podem ser

[2]*Plug and play*: termo que faz referência ao processo de conexão e posterior funcionamento de um dispositivo em um determinado sistema, sem a necessidade de reconfigurações.

disponibilizados através da mesma interface com o NCAP. Este fato é importante para aplicações que precisam de elevadas taxas de aquisição de dados.

Atualmente há um grupo de trabalho desenvolvendo a especificação IEEE P1451.5 (*Wireless Communication Protocols and Transducer Electronic Data Sheets*) visando sua futura aprovação. O grupo de trabalho tem como objetivo a criação de uma especificação para direcionar as aplicações do padrão IEEE 1451 em ambientes de rede sem fio [8].

Há também uma proposta recente denominada de P1451.0, cujo objetivo é a criação de uma diretriz que sirva de base para todas as outras especificações da família 1451, por exemplo, a unificação das TEDS das especificações componentes [9], [10].

Outra proposta lançada em 2004 foi a IEEE P1451.6 (*A High-speed CANopen-based Transducer Network Interface for Intrinsically Safe and Non-intrinsically Safe Applications*) [10]. Neste caso propõe-se o uso de uma rede de alta velocidade baseada no sistema *CAN-open*[3] com diversos módulos contendo transdutores e a definição de uma camada de segurança no modelo de comunicação.

1.3 – Contextualização do Projeto

Nesta seção são expostos os objetivos e estratégias de ação do trabalho, assim como também, a motivação e a justificativa correspondente. Os métodos e os materiais empregados no desenvolvimento do trabalho serão abordados no Capítulo 7.

O projeto de um sistema de instrumentação distribuída requisita a aplicação de diversas técnicas e metodologias relacionadas com aspectos relevantes como aquisição, interfaceamento e comunicação. O sistema concebido, além de ser eficiente, deve ser flexível o suficiente para que possam ser feitas modificações futuras sem a necessidade de efetuar pesadas reconfigurações. De modo geral, essa flexibilidade deve abranger desde a aquisição de dados até o receptor final das informações, o que implica que as diferentes interfaces envolvidas no sistema de instrumentação devem ser simples e intuitivas.

No projeto de sistemas de instrumentação distribuída devem levar-se em conta diferentes pontos críticos. A primeira das considerações diz respeito à descrição dos tipos de transdutores e

[3]*CAN-open*: protocolo de comunicação aberto para aplicações industriais baseado em CAN (*Controller Area Network*) e introduzido pelo grupo CiA (*CAN in Automation*).

dispositivos que constituirão o sistema, além de descrever a interface física necessária para cada aplicação. Na etapa posterior, devem ser analisadas as possíveis redes ou barramentos de campo que possam satisfazer as necessidades do projeto e que melhor se adaptem às características dos dispositivos, a fim de que os transdutores conectados possam ter um ótimo desempenho. Este fato, em certas ocasiões, só pode ser resolvido mediante o emprego de soluções proprietárias. Finalmente, os algoritmos de aplicação devem ser desenvolvidos com base em um modelo de distribuição adequado.

1.3.1 - Objetivos da Tese

O objetivo deste trabalho é o desenvolvimento de um sistema de instrumentação distribuída, composto por um nó de rede em conformidade com o padrão de interfaceamento IEEE 1451, para conectar transdutores inteligentes em ambientes de rede, utilizando-se para tal fim, ferramentas padronizadas e de domínio aberto.

Como parte do nó IEEE 1451, implementou-se um Processador de Aplicação com Capacidade de Operar em Rede (NCAP), com base no padrão IEEE 1451.1, agindo como servidor, e um Módulo de Interface para Transdutores Inteligentes (STIM), em conformidade com o padrão IEEE 1451.2 [11].

O STIM foi desenvolvido integralmente através dos recursos da tecnologia de lógica programável, utilizando-se a linguagem padronizada VHDL (*VHSIC*[4] *Hardware Description Language*) para descrever e sintetizar as funcionalidades especificadas no padrão em dispositivos lógicos programáveis de uso geral, conforme feito no trabalho citado em [12].

Por seu turno, o *hardware* do NCAP foi implementado com um microcomputador, sendo que foram usados: uma placa de rede, a porta paralela e um gerenciador de protocolo, desenvolvido em VHDL. A funcionalidade do IEEE 1451.1, em relação com o ponto de vista *software*, desenvolveu-se através dos recursos da programação orientada a objetos, especificamente, a tecnologia Java, pois além de puramente orientada a objeto e possuir relação direta com a Internet, possibilita a reutilização de código eficiente e apresenta portabilidade entre diferentes plataformas [13].

[4] VHSIC (*Very High Speed Integrated Circuits*) é o nome de um grupo de trabalho criado pelo departamento de defesa americano nos finais da década de 1970, com a finalidade de criar uma nova geração de circuitos integrados.

O conjunto NCAP-STIM foi conectado a uma rede baseada no padrão IEEE 802.3, sendo que, através da rede, diferentes NCAPs clientes podem acessar a aplicação, para realizar monitoramento e atuação remota das variáveis envolvidas no processo.

1.3.2 - Motivação

Já faz algum tempo que grandes companhias como a *GM* e a *Boeing* começaram a reavaliar como suas tecnologias de rede poderiam ser padronizadas através de tecnologias e protocolos associados à Internet [14].

A utilização de padrões reconhecidos internacionalmente, para implementar as interfaces e comunicações, constitui um ponto de importância vital no projeto de sistemas de instrumentação distribuída, escaláveis e flexíveis. Entretanto, na descrição de protocolos e interfaces podem ser utilizadas tanto ferramentas de domínio público quanto ferramentas proprietárias.

A utilização de ferramentas abertas possibilita ao usuário, empregá-la e até modificá-la, sem depender do projetista original, salientando-se que, esse recurso pode ser tanto *hardware* quanto *software*. Já uma ferramenta proprietária está associada a um determinado fabricante e sua aplicação pode fazer com que os sistemas desenvolvidos percam flexibilidade e interoperabilidade. Geralmente, um recurso proprietário mostra-se muito eficiente, porém, em virtude de sua natureza, resulta pouco flexível. Uma ferramenta de domínio público pode apresentar certas deficiências, entretanto, seu emprego possibilita que o sistema desenvolvido possa ser modificado e melhorado constantemente, introduzindo uma grande flexibilidade no projeto.

Fazendo-se uma análise em torno da flexibilidade de um sistema de instrumentação distribuída, surgem três conceitos fundamentais: *plug and play*, conectividade e interoperabilidade. O *plug and play* refere-se à possibilidade de evitar as dificuldades de reconfigurar um sistema, toda vez que é acrescentado um novo dispositivo ao mesmo. A conectividade refere-se à integração de diferentes equipamentos e dispositivos através de tecnologias de rede, enquanto que a interoperabilidade é a possibilidade de um determinado elemento *hardware* ou *software* operar com diferentes tecnologias.

O uso de soluções proprietárias pode acarretar, em algum grau, a perda de flexibilidade e dos conceitos de *plug and play* e interoperabilidade. Deve-se levar em conta que nos sistemas

distribuídos de medição e controle torna-se fundamental interconectar diferentes dispositivos e que nem todos eles são fornecidos pelo mesmo fabricante.

No Capítulo 4 será analisado o fato de existirem diversas metodologias de conexão possíveis e uma ampla variedade de tecnologias de rede e barramentos de campo disponíveis no mercado atual, a tal ponto que no final da década de 1980 surgiu o termo "guerra dos barramentos de campo" (*fieldbus war*) fazendo referência a esse problema [15]. Desta maneira, surge uma pergunta muito difícil de responder: qual a tecnologia mais conveniente para uma aplicação em particular?

Quando se deseja conectar um barramento de campo com uma rede tipo Intranet, compatível com *Ethernet*, é necessário desenvolver um *gateway* completo, para relacionar o protocolo do lado do barramento de campo com o protocolo do lado da rede. Geralmente, o protocolo utilizado pelo barramento é específico, sendo que o protocolo do lado da Intranet oferece várias alternativas. Evidentemente, a interconexão de transdutores à Internet/Intranet, mediante barramentos de campo, resulta em uma ampla variedade de soluções proprietárias. Contudo, existe a possibilidade de direcionar este tipo de problema, através do padrão de interfaceamento IEEE 1451.

Pelo fato de especificar um nó de rede com capacidade de processamento local e capacidade de interoperar com diferentes tecnologias de rede, implementando um sistema em conformidade com o padrão IEEE 1451, é possível pensar na conexão de transdutores inteligentes diretamente com redes do tipo local, por exemplo, *Ethernet*, utilizando o nó como dispositivo intermediário entre o transdutor e a rede.

Segundo Lee & Schneeman, em [4] e [16], a tendência atual em sistemas de medição e controle distribuído baseia-se na utilização de padrões para realizar o interfaceamento entre transdutores e redes, o emprego de padrões de comunicação e a utilização da *Web* para difundir as informações. Assim, os sistemas ficam muito mais flexíveis e integrados. Diferentes tecnologias associadas à Internet como a linguagem Java, WWW, o conjunto de protocolos TCP / IP e a *Ethernet*, estão se tornando as plataformas preferidas para construir a próxima geração de sistemas de medição e controle distribuídos.

Os mesmos autores, em [14], exploram técnicas com o objetivo de integrar as tecnologias associadas à Internet e os novos padrões do IEEE para transdutores inteligentes, com os sistemas de medição e controle distribuído. Desde esse ponto de vista consideram-se três importantes áreas

de integração: interfaces padronizadas para transdutores, redes abertas de comunicação e aplicações de modelos distribuídos.

Os fatos anteriores motivam a realização de pesquisas na área de padronização em sistemas de instrumentação. Como ponto inicial, no presente trabalho, é de particular interesse a implementação de um sistema baseado no padrão de interfaceamento IEEE 1451 com ênfase na utilização de ferramentas padronizadas e de domínio aberto para implementá-lo. Salientando-se que, explora-se também, a utilização do padrão IEEE 802.3 no nível de rede de controle.

1.3.3 - Justificativa

Os dispositivos transdutores fazem parte de um mercado de produtos muito amplo e variado, esperando-se um crescimento deste mercado, equivalente a US$ 43 bilhões até o ano de 2008 [17]. No entanto, de acordo com um estudo recente realizado pelo grupo *Itechno Consulting* da Suíça, o crescimento do mercado poderia superar os US$ 50 bilhões em 2008, sem levar em conta as aplicações militares [18]. Outro dado interessante é que ao ser considerado o mercado M2M (*Machine to Machine*), ou seja, tudo aquilo que abrange desde o transdutor até o receptor final das informações como, por exemplo, as tecnologias de rede e desenvolvimento de *software*, o valor poderia atingir os US$ 400 bilhões em 2010.

A ausência de um padrão internacional de interfaceamento fez com que, durante vários anos, os engenheiros projetistas da área de instrumentação tivessem de empregar muito tempo e esforço para projetar as interfaces de transdutores com diferentes tecnologias de rede, incrementando-se, assim, o custo das aplicações.

Um sistema de instrumentação e controle distribuído projetado em conformidade com o padrão IEEE 1451 e a filosofia de sistemas abertos possibilita a introdução de conceitos como: flexibilidade, interoperabilidade, *plug and play* e reutilização de códigos.

Embora o projeto possa ser desenvolvido para uma aplicação específica, a relevância do mesmo está no emprego do padrão e não no tipo de processo envolvido. Utilizando os mesmos conceitos, o projeto pode ser ampliado e, ainda, levado para outras áreas, quase sem esforços adicionais.

Através da implementação de um sistema em conformidade com as diretrizes do padrão IEEE 1451, as indústrias poderão obter inúmeros benefícios, tais como: redução de custos de sistemas de instrumentação devido à redução de complexidade, maior facilidade e flexibilidade para

transmitir informações através de uma rede de comunicação e maior facilidade para ampliar os sistemas sem efetuar grandes configurações. Porém, a relevância não abrange apenas o setor industrial, mas também áreas como a biomédica, telemedicina e automação residencial.

1.4 - Trabalhos Relevantes na Área

Um trabalho que teve contribuições relevantes na área de sensores inteligentes em rede, data do ano de 1995 [19]. Como parte dessa pesquisa foi desenvolvido um protótipo de sensor inteligente conectado em rede. O trabalho teve como finalidade contribuir para três áreas bem definidas: a) padronização de interfaces, b) utilização de especificações de transdutores em formato eletrônico e c) execução de tarefas de gerenciamento através de nós de rede associados aos transdutores.

No ano de 1996, Lee e Schneeman, em [20], implementaram um módulo inteligente em conformidade com as especificações IEEE P1451.1 e IEEE P1451.2 que ainda estavam em processo de aprovação. Porém, o trabalho teve como finalidade exemplificar e divulgar os conceitos especificados nas diretrizes acima mencionadas.

No ano de 1998 foi desenvolvido um circuito integrado com tecnologia CMOS, implementando um STIM que integrava conversores A/D, D/A e um microcontrolador de 8 *bits* com 256 *bytes* de memória SRAM (*Static Random Access Memory*) e 10,5 *Kbytes* de memória *flash* EEPROM (*Electrically-Erasable Programmable Read-Only Memory*), com condições satisfatórias de desempenho [21], [22].

Em 1999 foi implementado um sistema de medição e controle distribuído com base em padrões internacionais. Este trabalho provou a viabilidade dos sistemas de instrumentação projetados em conformidade com os padrões IEEE 1451.1, IEEE 1451.2, IEEE 802 (tecnologias de redes LANs), além da utilização das tecnologias associadas à Internet como, WWW e protocolos TCP/IP [14].

Em 2000 foi implementado um módulo STIM, em conformidade com o padrão IEEE 1451.2, utilizando um sistema embarcado fornecido pela empresa *Analog Devices* [23], [24]. Esse sistema continha, na mesma placa, um núcleo processador baseado no microcontrolador 8051, além de outros componentes como memórias, conversores A/D e D/A e interfaces com microcomputador. Esse trabalho provou que é possível desenvolver módulos STIM utilizando tecnologias existentes no mercado.

Em 2000 foi implementado um nó IEEE 1451 completo STIM-NCAP, para funcionar em um ambiente de rede com o protocolo CAN *(Controller Area Network)* [25]. Neste caso, tanto o STIM como o NCAP foram implementados utilizando microcontroladores comerciais suportados por alguns periféricos e ambientes de *software*.

Em 2001 foi implementado um sistema de instrumentação experimental, integrando o padrão IEEE 1451, o padrão IEEE 802.3 e tecnologias associadas à Internet [26]. A funcionalidade especificada no padrão IEEE 1451.2 foi desenvolvida, em parte, utilizando a tecnologia de lógica programável. Já a funcionalidade especificada no padrão IEEE 1451.1 foi atingida através do emprego de placas comerciais.

Em 2002 foi desenvolvida uma aplicação IEEE 1451.1, para acessar sensores via Internet utilizando comunicação série e protocolo TCP/IP [27]. O NCAP neste caso foi implementado através de uma placa comercial [28].

Em 2002 foi desenvolvido um STIM de dois canais, utilizando a tecnologia de lógica programável [29]. O STIM foi implementado com dispositivo do tipo Matriz de Portas Programável em Campo (FPGA - *Field Programmable Gate Array*) de cerca de 800.000 portas típicas.

Em 2002 foi implementado, em laboratório, um sistema embarcado compatível com vários protocolos de comunicação [30]. O sistema desenvolvido implementava, em parte, um módulo STIM baseado na tecnologia de lógica programável e parte da funcionalidade especificada no padrão IEEE 1451.1, utilizando um microcontrolador.

Em 2004 foi proposto um NCAP simples baseado em microcontrolador e reprogramável dinamicamente através de uma rede local [31].

Em 2004 foi implementado um sistema IEEE1451, para monitoramento de níveis de CO_2 no ar, através de um STIM contendo um sensor infravermelho não dispersivo conectado à uma rede com características específicas, vinculada à Internet [32].

1.4.1 – Comparação entre diferentes implementações IEEE 1451

Na Tabela 1.1 é apresentada uma comparação entre os diferentes trabalhos referenciados na seção 1.4 e o presente trabalho de tese.

Tabela 1.1. Comparação entre diferentes implementações IEEE 1451.

Referência / Tipo de implementação	Recursos empregados para a implementação do NCAP	Recursos empregados para a implementação do STIM	Tecnologia de Rede	Características/ possíveis aplicações
[20]: Implementação demonstrativa	- PC - *Software* do NCAP em C++	- Sensor de pressão compatível com a especificação IEEE P1451.2 - TEDS *on-board*	- *Ethernet* 10BaseT	- Experimental
[21], [22]: STIM	-	- Circuito integrado baseado em microcontrolador de 8 *bits* - Tecnologia CMOS	-	- Experimental
[14]: Sistema de instrumentação distribuída	- Baseado em [20] e tecnologias diversas - *Software* em C, C++ e Java *Applet*	-	- *Ethernet* 10BaseT	- Experimental - Controle de temperatura
[23]: Aplicação do padrão IEEE 1451.2	- Solução Comercial - HP – BFOOT 65501 (*Hewlett Packard*)	- Sistema de aquisição AduC812 + *Kit* de desenvolvimento Keil C-51 (*Analog Devices*) - 2 Canais transdutores	- *Ethernet*	- Controle de temperatura
[25]: Nó IEEE 1451	- *Hardware* com núcleo uC 16HC9161X1 (*Motorola*) + Controlador CAN 82C250	- Microcontrolador 87C752 (*Philips*) - 2 Canais Transdutores	- CAN	- Indústria automotiva
[26]: Sistema de instrumentação	- Solução Comercial - HP – BFOOT 65501 (*Hewlett Packard*)	- FPGA *multicore* APEX 20K200 de 200.000 portas típicas (*Altera*)	- *Ethernet* - *Fast Ethernet*	- Experimental - Física nuclear
[27]: NCAP	- Solução Comercial EM04a (*Esensors*)	-	- *Ethernet* 10BaseT	- *Websensors*[5]
[29]: Aplicação do padrão IEEE 1451.2	-	- FPGA VIRTEX XCV800 de cerca de 800.000 portas (*Xillinx*) - 2 Canais Transdutores	-	- Gerais
[30] NCAP-STIM (SoC – *System-on-Chip*)	- Microprocessador do tipo 6502	- FPGA VIRTEX XCV800 (*Xillinx*)	-	- Gerais
[31]: NCAP	- *Hardware* com núcleo uC MCS51	-	- Rede LAN	- Gerais
[32]: Nó IEEE 1451	- Microcontrolador	- Microcontrolador - Sensor infravermelho para monitoramento de CO_2	- Rede específica (emNET)	- Monitoramento de níveis de CO_2 no ar
Trabalho de Tese	- PC convencional + gerenciador - de protocolo (FPGA FLEX EPF10K20RC240-4 de 20.000 portas típicas, *Altera*) - *Software* do NCAP em Java	- FPGA ACEX EP1K50TC144-3 de 50.000 portas típicas (*Altera*) - 2 Canais Transdutores	- *Ethernet* 10BaseT	- Gerais

[5] *Websensor*: sistema sensor que pode ser acessado diretamente desde um *site* na Internet.

Como mencionado na Tabela 1.1, em [20], foi implementado um NCAP-PC utilizando a porta paralela. No entanto, pelo fato do padrão IEEE 1451.2 definir uma interface serial padronizada composta por dez sinais, o *driver* da porta paralela teve de ser modificado. Evidentemente essa solução faz com que a portabilidade de uma determinada aplicação seja perdida. No presente trabalho escolheu-se um PC para implementar o NCAP, entretanto foi desenvolvido um dispositivo gerenciador de protocolo que realiza parte da funcionalidade 1451, possibilitando sua conexão com qualquer PC. O PC deve executar o *software* do NCAP que foi desenvolvido, neste trabalho, por meio da linguagem Java de domínio público. Em [20] o *software* do NCAP foi desenvolvido em C e C++.

Na alternativa apresentada em [21] e [22] foi desenvolvido um circuito integrado com base em um núcleo microcontrolador de 8 *bits*, visando a implementação de um STIM experimental.

No STIM implementado em [23] utilizou-se um sistema embarcado fornecido pela empresa *Analog Devices*. Outra solução comercial apresenta-se em [27], com base em um *kit* de desenvolvimento para aplicações IEEE 1451 fornecido pela empresa *Esensors*.

Além de ter-se notado uma carência de aplicações de baixo custo, existem poucas soluções baseadas em ferramentas abertas e padronizadas. Desta maneira, optou-se por implementar o STIM integralmente com dispositivos lógicos programáveis versáteis e de baixo custo, contendo dois canais transdutores, pois a tecnologia de lógica programável possibilita o projeto de sistemas digitais flexíveis e a reutilização de códigos através da linguagem padronizada VHDL.

Em [26] o STIM foi implementado com uma FPGA *multicore*[6], que possibilita a implementação de sistemas digitais complexos, porém, o custo desses dispositivos é maior do que o custo dos dispositivos de lógica programável de uso geral e, além disso, seu uso torna-se mais difícil. Em [29], utilizou-se uma FPGA de 800.000 portas típicas, para um STIM com dois canais transdutores. Em contraste, no presente trabalho fez-se uma avaliação com três FPGAs versáteis e de uso geral de 20.000, 50.000 e 100.000 portas típicas, sendo que um STIM com dois canais transdutores pode ser implementado com uma FPGA de 50.000 portas típicas modelo EPF1K50TC14-3 da família ACEX da empresa *Altera*.

[6]*Multicore*: sistema que combina diferentes estruturas de dispositivos lógicos programáveis em uma única arquitetura, eliminando assim, a necessidade de diversos componentes.

Em [25], [30], [31], [32] foram utilizadas soluções comerciais e microcontroladores. Embora o emprego de microcontroladores seja uma solução de baixo custo, possui uma desvantagem importante. Geralmente a utilização de microcontroladores requer o uso de linguagem *assembly* que é específica de cada fabricante, fazendo com que as aplicações fiquem fechadas e, por conseguinte, percam flexibilidade.

Podem ser citados outros trabalhos realizados no NIST como, por exemplo, alguns testes experimentais feitos com tecnologias de rede para aplicações industriais como: *LonWorks, DeviceNet* e *Smart Distributed System* (SDS) [17].

Resumindo as idéias expostas, a contribuição global do presente trabalho é o emprego de ferramentas padronizadas e abertas na implementação de um nó IEEE 1451. Entretanto, há várias contribuições pontuais, como o emprego de FPGAs versáteis e de uso geral para implementar o STIM, a maneira com que foram implementadas as TEDS, o desenvolvimento de um gerenciador de protocolo em VHDL com o objetivo de tornar portável o uso de um NCAP-PC e o desenvolvimento do *software* do NCAP integralmente na linguagem Java. Estas idéias serão tratadas com mais detalhes nos Capítulos 7 e 8.

1.5 - Organização do Texto

Incluída a introdução geral correspondente ao Capítulo 1, que contém a contextualização do projeto e os trabalhos mais relevantes na área, o texto foi organizado em nove capítulos.

No Capítulo 2 são introduzidos os principais conceitos associados às redes de computadores, sendo que muitos deles foram aplicados diretamente e outros têm a finalidade de esclarecer ao leitor sobre a terminologia e os conceitos utilizados no padrão de IEEE 1451. Nesse Capítulo abordam-se também, alguns aspectos das redes industriais.

No Capítulo 3, que trata os sistemas de instrumentação distribuída, são abordados tópicos sobre instrumentação, dispositivos transdutores e sistemas de medição e controle distribuídos.

No Capitulo 4 trata-se a conexão de transdutores em rede, abordando-se em detalhe: técnicas, metodologias e tecnologias disponíveis comercialmente para este fim. Na seção 4.2.3 será apresentado o tópico: redes e barramentos de campo comerciais, cuja finalidade é ilustrar a diversificação de tecnologias para conectar transdutores em ambientes de rede.

Na seqüência aborda-se, no Capítulo 5, o padrão de interfaceamento IEEE 1451, visando o esclarecimento dos seus objetivos e suas contribuições na área de instrumentação distribuída. Em particular trata-se o padrão IEEE 1451.1 e o padrão IEEE 1451.2, cujas especificações foram usadas para a implementação do nó de rede, motivo do presente trabalho de tese.

No Capítulo 6 são introduzidas as tecnologias empregadas no projeto, começando pela *Ethernet*, utilizada em nível de rede de controle, e na seqüência, a tecnologia de lógica programável e tecnologia Java utilizadas no desenvolvimento do nó IEEE 1451. Apresenta-se um estudo sobre limitações e vantagens de cada uma delas e sua influência no desempenho do nó implementado.

No Capítulo 7 são apresentados os materiais e os métodos empregados visando atingir os objetivos mencionados na seção 1.3.1, abordando-se seqüencialmente o desenvolvimento do STIM, do NCAP e da conexão do nó na rede.

No capítulo 8 são apresentados os resultados de simulação e os resultados experimentais com a correspondente análise.

Finalmente, no Capítulo 9, são apresentadas as conclusões gerais da tese. Além disso, o trabalho contém as referências utilizadas, um glossário e um Apêndice dividido em seis seções.

Capítulo 2

Redes de Computadores

O presente capítulo tem como objetivo introduzir os principais conceitos associados às redes de computadores que, de maneira direta ou indireta, têm sido utilizados neste trabalho. No final do capítulo abordam-se aspectos básicos das redes industriais.

2.1 - Definição e Objetivos das Redes de Computadores

Em meados da década de 1960, a idéia de computador centralizado foi mudando gradativamente para dar passo ao conceito de computadores em rede. O fator custo foi uma das principais causas da migração dos sistemas computadores de grande porte para as redes de computadores, pois, a relação preço/desempenho dos pequenos computadores mostrava-se muito melhor.

Define-se rede de computadores como sendo um conjunto de computadores autônomos interconectados que podem trocar informações. A conexão dos mesmos pode ser feita de diferentes maneiras, através de fios de cobre, fibras ópticas, técnicas de microondas e satélites.

Compartilhar recursos é o principal objetivo das redes de computadores. Através delas é possível colocar programas**,** dispositivos e dados ao alcance de todas as pessoas da rede [33].

Uma rede de computadores aumenta a confiabilidade de um sistema de informação, uma vez que ela possui fontes alternativas de fornecimento e oferece escalabilidade, que é a possibilidade de aumentar gradualmente o desempenho de um sistema à medida que cresce o volume de carga, bastando, para tal, que se adicionem mais processadores.

2.2 - Classificação das Redes por Escala

De acordo com a abrangência geográfica, as redes de computadores podem ser classificadas em redes de área local, redes metropolitanas, redes geograficamente distribuídas e inter-redes. Na Tabela 2.1 mostra-se uma classificação detalhada.

Tabela 2.1. Classificação das redes por escala.

Distância do interprocessador	Processadores no(a) mesmo(a)	Exemplo
0.1 m	Placa de circuitos	Máquina de fluxo de dados
1 m	Sistema	Multicomputador
10 m 100 m 1 Km	Sala Prédio Campus	Rede Local (LAN)
10 Km	Cidade	Rede Metropolitana (MAN)
100 Km 1000 Km	País Continente	Rede Geograficamente Distribuída (WAN)
10000 Km	Planeta	Inter-Rede

Fonte: Tanenbaum, A.S. Redes de Computadores. Adaptado.

2.2.1 - Rede de Área Local (LAN)

As LANs são redes contidas em um prédio, empresa, universidade, etc. Os componentes principais de uma rede LAN são os servidores, as estações de trabalho (clientes) e os recursos (programas, impressoras, etc). Em geral, as LANs operam a velocidades que podem variar de 10 a 100 ou mais Mbps[7]. As tecnologias de redes de área local estão em conformidade com o padrão IEEE 802[8]. A tecnologia de rede local mais utilizada na atualidade obedece ao padrão IEEE 802.3, comumente conhecido como *Ethernet* [34]. Na Figura 2.1 são mostradas duas topologias básicas de LANs. A topologia de rede faz referência à organização do cabeamento do sistema.

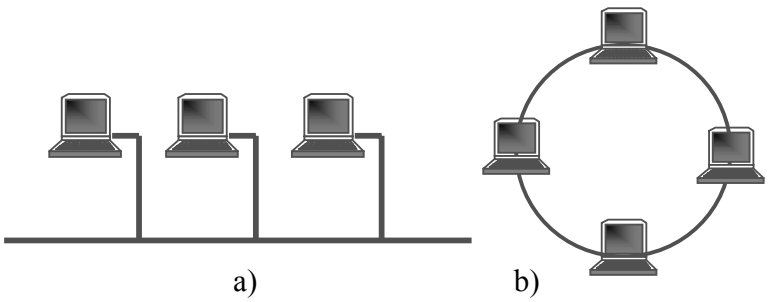


Figura 2.1. Duas topologias de redes LAN, a) Barramento; b) Anel

[7] Mbps: unidade de medida de largura de banda usada para expressar um milhão de *bits* por segundo.
[8] Para maiores informações sobre o padrão IEEE 802 pode recorrer-se ao Apêndice A.

a) Topologia em Barramento

Na topologia em barramento todas as máquinas são conectadas ao mesmo meio físico, fazendo com que as mensagens transmitidas através do cabo sejam recebidas por todas as máquinas da rede. Esse modo de operação é chamado de difusão (*broadcasting*) [33]. Esta topologia é de fácil implementação, entretanto, o aumento da distância pode degradar os sinais transmitidos.

b) Topologia em Anel e Topologia em Estrela

Nesta topologia, os dispositivos são conectados a um sistema de cabeamento que forma um círculo ou anel, como mostrado na Figura 2.1 b. As mensagens circulam em uma só direção e são lidas por todos os computadores e retransmitidas ao anel no caso daquele computador não ser o destinatário final da informação. O exemplo típico de rede que utiliza topologia em anel é a *Token Ring* da IBM, padronizada através do IEEE 802.5. Nela, uma ficha (*token*) circula pela rede e, caso ela esteja vazia, um computador pode "completar" a ficha e enviar um quadro de dados para outra máquina na rede [35]. A rede *Token Ring* é implementada com uma topologia lógica em anel e uma topologia física em estrela, por questões de flexibilidade ao inserir novas máquinas na rede, pois a interrupção de um anel físico bloquearia a rede. Na Figura 2.2, é mostrada a topologia em estrela, utilizando um concentrador, que o dispositivo responsável por implementar o anel lógico.

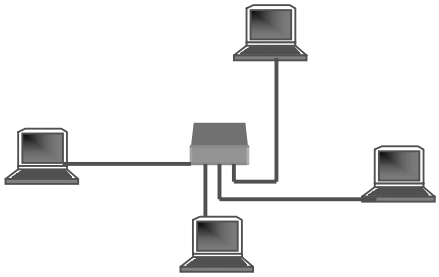


Figura 2.2. Topologia em estrela.

2.2.2 - Rede de Área Metropolitana (MAN)

As MAN são uma versão ampliada das LANs, pois basicamente os dois tipos de rede utilizam tecnologias semelhantes. Uma MAN pode abranger um grupo de escritórios vizinhos ou uma

cidade. O fato das MANs serem tratadas como uma categoria especial deve-se a que elas obedecem a um padrão em particular, o IEEE 802.6.

2.2.3 - Redes Geograficamente Distribuídas (WAN)

As WANs possuem uma maior abrangência geográfica, podendo abranger desde um país até um continente. Geralmente, as WANs interconectam LANs distantes geograficamente utilizando linhas de transmissão fornecidas por empresas de telecomunicações.

2.2.4 - Inter-Redes

Normalmente, usuários distantes por milhares de quilômetros, conectados a diferentes redes, precisam comunicar-se entre si. Nesses casos é preciso que se estabeleçam conexões entre redes que podem ser incompatíveis. Às vezes, isso só é possível através da utilização de equipamentos chamados de *gateways*, também conhecidos como conversores de protocolo. Assim, um conjunto de redes interconectadas é chamado de inter-rede. Um exemplo específico é a Internet, de alcance mundial.

2.3 - Camadas, Protocolos, Interfaces e Serviços

Com o objetivo de reduzir o grau de complexidade, a estrutura de *software* de uma rede divide-se em diferentes camadas. Deste modo, a camada n de uma máquina ou *host*[9] se comunica com a camada n de outra, a fim de lhe oferecer determinados serviços. O conjunto de regras sobre o modo como se estabelecerá a comunicação entre as partes envolvidas denomina-se de protocolo de camada n [33]. Na Figura 2.3 é ilustrado o conceito de forma esquemática.

Na verdade, os dados não são transferidos diretamente desde a camada n de uma máquina para a camada n da outra. A informação percorre os diferentes níveis até atingir a camada requisitada. Nesse caminho, as informações passam pelo meio físico, através do qual se dá a comunicação propriamente dita.

[9] *Host:* termo utilizado para fazer referência a um sistema computador conectado a uma rede. Similar ao termo nó de rede.

Entre cada par de camadas adjacentes existe uma interface. A interface define as operações e serviços que uma camada tem a oferecer para a outra. O conjunto de camadas, protocolos e interfaces, denomina-se arquitetura de rede.

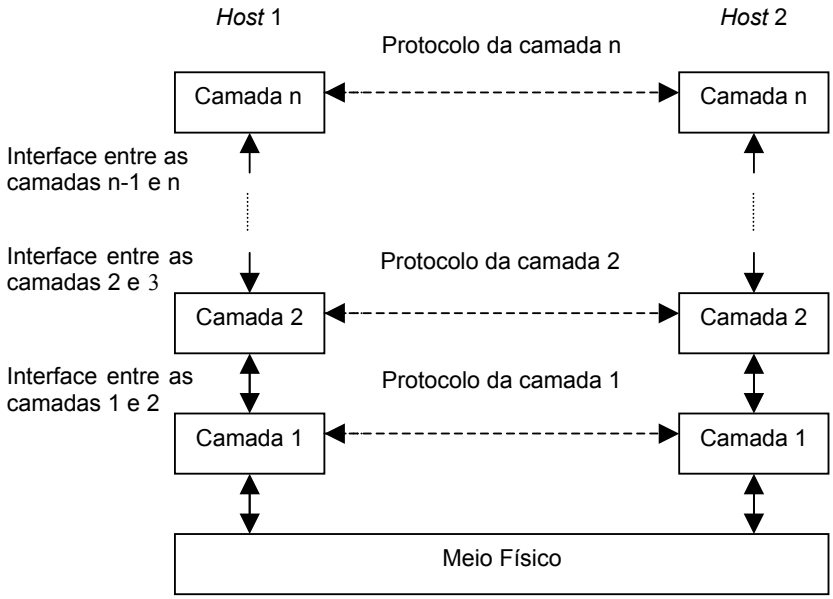


Figura 2.3. Camadas, protocolos e interfaces.

Quanto aos serviços, existem serviços orientados à conexão e serviços sem conexão ou de datagrama. O serviço orientado à conexão confiável, é baseado no sistema telefônico. O usuário estabelece uma conexão, utiliza a conexão e finalmente a libera. O emissor envia *bits* e o receptor os recebe na ordem em que foram enviados. Neste tipo de serviço pode-se dispor de confirmação, mas, os retardos introduzidos pelas confirmações podem ser simplesmente inaceitáveis. O serviço sem conexão, ou de datagrama, é baseado no sistema postal. Cada mensagem carrega o endereço de destino completo e cada um deles é roteado através do sistema independentemente de todos os outros. O serviço de datagrama é usado, por exemplo, para o tráfego de voz digital ou a transmissão de um vídeo, quando não há problema se aparecem alguns *bits* de ruído ou alguns *pixels* errados durante a transmissão [33].

Um serviço é especificado por um conjunto de primitivas disponíveis para que uma entidade possa acessá-lo. Por exemplo, a primitiva *request* que faz referência a uma entidade querendo que o serviço execute alguma ação, ou a primitiva *response*, fazendo referência a uma entidade querendo responder a um evento.

2.4 - Transmissão de Dados

A transferência de dados na rede pode-se dar através de alguma das seguintes formas:
a) Transferência em apenas uma direção ou comunicação simplex.
b) Transferência em duas direções, mas não simultaneamente ou comunicação *half duplex*.
c) Transferência em duas direções simultaneamente ou comunicação *full duplex*.

2.4.1 -Transmissão Paralela

Em termos genéricos, na comunicação paralela o transmissor envia ao mesmo tempo para o receptor, todos os *bits* que ele é capaz de transmitir, por exemplo, uma palavra de oito *bits*. A transmissão paralela é utilizada internamente em processadores e computadores [35]. O meio físico precisa de um fio por *bit* transmitido e, portanto, um fio pode causar interferência no seu adjacente. Outro fenômeno que pode aparecer na transmissão paralela é o *skewing*, que está relacionado com a diferença entre os tempos de propagação dos diferentes sinais através do meio de transmissão. Assim, a comunicação paralela depende do meio físico usado para a transmissão e o comprimento dos cabos não deve passar de uns poucos metros. A transmissão paralela é mais rápida do que a série, que será abordada a seguir.

2.4.2 - Transmissão Série

Na transmissão em série o transmissor envia os *bits* um a um precisando apenas de um fio para tal fim. Em geral, a comunicação em série é mais lenta do que a paralela. Contudo, podem se levar em conta distâncias maiores. Por este motivo, as redes locais utilizam este tipo de transmissão.

2.5 - Modelos de Referência

Existem dois modelos de referência mundialmente conhecidos, o modelo ISO/OSI e o modelo TCP/IP. Embora estes modelos sejam chamados muitas vezes de arquiteturas de rede, não são estritamente arquiteturas de rede, pois apenas fazem referência a camadas e não a protocolos e interfaces. A seguir abordam-se ambos modelos.

2.5.1 - Modelo de Referência ISO/OSI

O modelo de referência ISO/OSI surgiu a partir de uma proposta desenvolvida pela Organização de Padrões Internacionais (ISO - *International Standards Organization*). A idéia era que os fabricantes pudessem criar protocolos baseados nesse modelo. A sigla OSI, do inglês *Open Systems Interconnection*, significa interconexão de sistemas abertos, ou seja, conexão de sistemas de diferentes fabricantes, mas que obedecem ao mesmo padrão. Apesar de que, a maioria dos protocolos existentes não segue o modelo de maneira estrita, o modelo constitui uma ferramenta didática essencial, pois através dele é possível entender o funcionamento do protocolo ideal. Na Figura 2.4 são mostradas as sete camadas do modelo OSI, que são descritas a seguir.

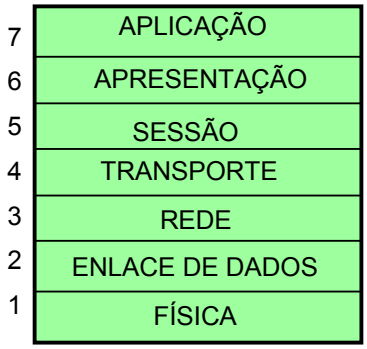


Figura 2.4. Modelo de referência ISO/OSI.

a) Camada de Aplicação

A camada de aplicação é a interface entre o protocolo de comunicação e o aplicativo, que lhe solicita para executar alguma tarefa através do usuário.

b) Camada de Apresentação

A camada de apresentação converte o formato de dado recebido pela camada de aplicação em um formato comum entendido pelo protocolo de comunicação para ser usado nas outras camadas. Portanto, a camada de apresentação se preocupa com a sintaxe e a semântica das informações transmitidas.

c) Camada de Sessão

Esta camada possibilita que os usuários de diferentes máquinas estabeleçam sessões. Na sessão mencionada, as aplicações definem o modo em que será feita a comunicação e colocam marcadores nos dados transmitidos. Se a rede falhar, os computadores reiniciarão a transmissão de dados a partir da última marcação recebida pela máquina receptora.

d) Camada de Transporte

A função básica da camada de transporte é aceitar dados da camada de sessão, dividi-los em unidades menores e passá-los para a camada de rede. Do lado do receptor, a camada de transporte é responsável por pegar os pacotes recebidos da camada de rede e construir o dado original para enviá-lo à camada de sessão.

e) Camada de Rede

A camada de rede diz respeito ao modo como os pacotes de informação serão roteados da origem para o destino. Fala-se em *rota* quando a rede contém mais de um segmento, havendo, portanto, mais de um caminho para um pacote de dados trafegar da origem até o destino. Quando um pacote tem de viajar de uma rede para outra até chegar ao nó de destino, podem surgir diferentes problemas. Talvez a rede de destino não aceite o tamanho do pacote enviado ou talvez utilize um protocolo diferente. Caberá à camada de rede resolver esses problemas, fazendo com que redes heterogêneas possam se comunicar.

f) Camada de Enlace de Dados

Sua tarefa principal é pegar os pacotes de dados recebidos da camada de rede e transformá-los em quadros a serem trafegados pela rede, acrescentando informações como o endereço da placa de rede de origem, o endereço da placa de rede de destino, dados de controle, os dados propriamente ditos e o CRC[10]. Na Figura 2.5 é mostrado um formato de quadro genérico.

[10] CRC (*Cyclical Redundancy Check*): Operação que consiste em somar todos os *bytes* de um pacote e enviar o resultado dentro do próprio pacote. A placa de rede do receptor verificará este resultado.

*	Identificadores	Dados	Detecção / Correção de Erros	*

*: bits de sincronismo.

Figura 2.5. Formato de quadro genérico.

g) Camada Física

O quadro de dados criado pela camada de enlace de dados é enviado para a camada física para serem convertidos em sinais elétricos e enviados através do cabo da rede. Os aspectos mais comuns são: o valor (em volts) para representar um *bit* 1 e um *bit* 0, o tempo (em microsegundos) que um *bit* deve durar, o fato da transmissão ser ou não realizada em duas direções, a forma como a conexão inicial será estabelecida e de que maneira ela será encerrada e a quantidade de pinos que precisará o conector da rede. O papel desta camada é desempenhado pela placa de rede.

2.5.2 - O Modelo de Referência TCP/IP

Em meados da década de 1960 o departamento de defesa dos Estados Unidos criou uma rede WAN de pesquisa chamada ARPANET, que constituiu o ponto inicial da Internet. No começo da década de 1970 o crescimento da ARPANET forçou a criação de um novo modelo de referência. Esse modelo, mostrado na Figura 2.6, veio ficar conhecido como modelo de referência TCP/IP (*Transmission Control Protocol*/*Internet Protocol*), graças a seus dois principais protocolos constituintes. As camadas componentes do modelo são descritas a seguir [33].

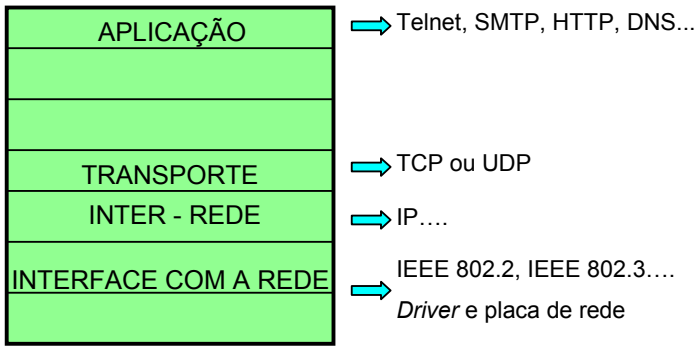


Figura 2.6. Modelo de referência TCP/IP.

a) Camada de Aplicação

Como mostrado na Figura 2.6, a camada de aplicação do modelo TCP/IP equivale às camadas 5, 6 e 7 do modelo OSI e executa a comunicação entre os aplicativos e o protocolo de transporte. Na camada de aplicação operam diferentes protocolos de alto nível como:

a.1) Telnet: protocolo de terminal virtual para efetuar *login* remoto em uma máquina de uma rede.

a.2) *Simple Mail Transfer Protocol* (SMTP): protocolo de correio eletrônico.

a.3) *File Transfer Protocol* (FTP): protocolo de transferência de arquivos entre *hosts* conectados à rede.

a.4) *HyperText Transfer Protocol* (HTTP): protocolo de transferência padrão da *Web*.

a.5) *Domain Name System* (DNS): protocolo utilizado para identificação de máquinas através de nomes em vez de endereços IP.

A camada de aplicação comunica-se com a camada de transporte através de portas. As portas são identificadas através de números e as aplicações padrão usam sempre uma mesma porta. Por exemplo, o protocolo HTTP utiliza sempre a porta 80 e o SMTP, a porta 25.

Outro aspecto importante do TCP é o conceito de *socket*. O *socket* é uma estrutura de *software* que vincula pontos finais de comunicação entre dispositivos de rede. O *socket* está associado com um número de porta e com o endereço do *host* na rede.

b) Camada de Transporte

Sua finalidade é pegar os dados enviados pela camada de aplicação e transformá-los em pacotes, a serem passados para a camada de rede, da mesma forma como acontece na camada de transporte OSI.

Nesta camada operam dois protocolos, o primeiro deles, o TCP é um protocolo orientado à conexão confiável que permite a entrega sem erros de um fluxo de *bytes* originado de uma máquina determinada, em qualquer nó da inter-rede. O TCP divide o fluxo de *bytes* de entrada em mensagens e passa cada uma delas para a camada inter-rede. Outra tarefa do protocolo TCP é impedir que um transmissor rápido sobrecarregue um receptor lento com um volume de mensagens muito grande.

Outro protocolo que opera na camada de transporte é o *User Datagram Protocol* (UDP). O UDP é um protocolo sem conexão, não confiável, para aplicações que não necessitam nem de controle de fluxo, nem de manutenção da seqüência das mensagens enviadas. Ele é amplamente utilizado em aplicações onde a entrega imediata é mais importante do que a entrega precisa, como a entrega de dados de voz ou de vídeo.

c) Camada Inter-Redes

Sua tarefa é permitir que os *hosts* injetem pacotes em qualquer rede e garantir que eles sejam transmitidos independentemente do destino. É possível que os pacotes cheguem em uma ordem diferente daquela em que foram enviados, obrigando as camadas superiores a reorganizá-los.

Deve-se observar que, nesse caso, a expressão inter-rede é utilizada no sentido genérico, muito embora essa camada esteja presente na Internet.

A camada inter-redes define um formato de pacote oficial e um protocolo chamado de *Internet Protocol* (IP). A tarefa da camada inter-redes é, então, entregar pacotes IP onde eles são necessários. O roteamento de pacotes é uma questão de grande importância nessa camada, assim como evitar congestionamentos.

d) Camada de Interface com a Rede

Esta camada equivale às camadas 1 e 2 do modelo OSI e sua função é a de enviar os pacotes recebidos pela camada de inter-rede em forma de um quadro através da rede.

2.5.3 - Modelo ISO/OSI vs. TCP/IP

O modelo OSI introduz três conceitos fundamentais: serviços, interfaces e protocolos. Provavelmente a maior contribuição do modelo OSI é tornar explícita a distinção entre esses três conceitos. Em contrapartida, o modelo TCP/IP não distinguiu com clareza a diferença entre serviço, interface e protocolo. Mas, ao contrário do que aconteceu com o modelo OSI, no TCP/IP, os protocolos vieram primeiro e o modelo foi criado com base neles. Os protocolos não tiveram problemas para se adaptar ao modelo [33].

A diferença fundamental entre os modelos é o número de camadas utilizadas. É possível constatar também que a maioria das tecnologias de rede existentes no mercado, não utiliza as

camadas de sessão e apresentação do modelo OSI. Tanenbaum, em [33], considera um modelo híbrido entre o modelo OSI e o TCP/IP baseado em 5 camadas, dispensando as camadas de apresentação e sessão, justificando seu uso com base em sua maior praticidade. As redes de campo para aplicações industriais utilizam geralmente as camadas física, enlace de dados, transporte e aplicação.

2.6 – Redes Industriais

Até o inicio da década de 1980, a arquitetura de gerenciamento das informações na maioria das empresas operava através de computadores de grande porte, oferecendo terminais aos quais os funcionários podiam se conectar [33]. Porém, seu elevado custo fez com que começassem a se tornar inviáveis.

O início da década de 1980 caracterizou-se pelo aparecimento do Computador Pessoal (PC - *Personal Computer*) da IBM e um grande desenvolvimento de *software* para aplicações em controle e supervisão. Com o transcorrer dessa década, começaram também, as aplicações de microprocessadores em ambientes industriais e apareceram os primeiros sensores inteligentes. Somado a esses fatos, a diminuição de custos dos equipamentos e dos circuitos integrados motivou a introdução do conceito de rede em ambientes industriais, possibilitando a coexistência de diversos equipamentos e dispositivos para o controle e monitoramento de processos, para o controle de máquinas e acionamentos e para o processamento de informações em nível de gestão.

2.6.1 - Definição e Objetivos das Redes Industriais

Uma rede industrial é um conjunto de equipamentos e dispositivos autônomos interconectados que podem trocar informações em diferentes níveis. A rede pode ser composta por computadores de projeto e gerenciamento, interfaces homem-máquina, Controladores Lógicos Programáveis (PLC – *Programmable Logic Controller*) e transdutores.

A redução de custos de instalação e o compartilhamento de recursos são os principais objetivos de uma rede industrial. Além disso, a sua implementação pode impulsionar o avanço significativo nos seguintes aspectos:

a) Manutenção;

b) Opções de atualização;

c) Informações de controle e qualidade;

d) Monitoramento de um processo produtivo completo;

e) Acesso e monitoramento remoto das variáveis de um processo.

2.6.2 - Níveis em uma Rede Industrial

A forma típica de agrupar os equipamentos e dispositivos em uma estrutura de automação industrial, ainda hoje, é a denominada pirâmide organizacional onde são estabelecidos diferentes níveis ou hierarquias [36]. Em geral, essas hierarquias são chamadas de nível de gestão ou nível de informação (o mais elevado), nível de controle, nível de campo e nível de entradas e saídas (o mais baixo). Na Figura 2.7, ilustra-se a estrutura piramidal de forma esquemática.

Figura 2.7. Diferentes níveis em uma rede industrial.

No nível de informação da rede utiliza-se geralmente um *software* de gerenciamento, implementando-se uma rede do tipo LAN ou WAN, por exemplo, para integrar uma rede de plantas pertencentes a uma indústria. O padrão IEEE 802 para redes locais, operando com o protocolo TCP/IP é a alternativa mais utilizada neste nível. O nível intermediário denomina-se de controle da rede. Isto é, a rede central localizada em uma planta, que incorpora, dentre outros dispositivos, CLPs e PCs. Neste nível, a informação deve mover-se em tempo real a fim de garantir a atualização de dados no *software* encarregado da supervisão da aplicação. O nível mais baixo refere-se geralmente às conexões físicas da rede ou barramentos de campo e às entradas e saídas, ou seja, aos sensores e atuadores. Quanto à utilização de barramentos de campo, as opções

são muito variadas, por exemplo: *Foundation Fieldbus*, *Profibus*, *WorldFip*, *Interbus*, *CAN*, *Sercos*, etc. Este tópico será abordado no capítulo 4.

2.6.3 - Tendências

Cabe considerar que a tendência em automação industrial indica a passagem da estrutura piramidal clássica para estruturas mais planas e integradas (*flat*) [36]. O protocolo TCP/IP é o grande candidato a se tornar o vínculo entre o nível de informação e o nível de campo. Existem dois motivos muito fortes para argumentar este fato: a) a utilização do controle baseado em PC e b) a presença da Internet cada vez mais acentuada no âmbito industrial.

A crescente necessidade de disponibilizar dados *on-line* e a utilização de interfaces que permitam ao usuário acessar as informações desde diferentes *hosts*, em diferentes locais, convergem para a utilização de um protocolo padrão, o TCP/IP. Assim é possível imaginar um novo modelo de estrutura organizacional integrado, sendo o protocolo TCP/IP o meio principal de transporte da informação.

2.7 – Comentários Finais sobre o Capítulo 2

Como parte do trabalho de tese devem ser conectados transdutores em ambiente de rede, portanto, muitos dos conceitos apresentados neste capítulo serão empregados no desenvolvimento do trabalho de tese. A terminologia apresentada será amplamente utilizada nos capítulos sucessivos, em particular, no capítulo 6 onde será abordada tecnologia *Ethernet*, que foi empregada no projeto, em nível de rede de controle.

Embora os conceitos abordados no trabalho possam ser empregados em diferentes áreas, provavelmente, o setor mais representativo seja o industrial. Em função disso, é importante ter em mente os aspectos apresentados sobre tendências em redes industriais.

Capítulo 3

Sistemas de Instrumentação Distribuída

O presente trabalho insere-se no contexto dos sistemas de instrumentação distribuída. Neste capítulo serão abordados tópicos elementares relacionados com os sistemas de medição e controle distribuído, abrangendo-se desde o conceito de transdutor até a necessidade de se implementarem sistemas distribuídos. Os conceitos aqui estudados são de importância vital na metodologia de implementação deste trabalho.

3.1 - Introdução

Em todo processo industrial, os dispositivos transdutores desempenham um papel de importância vital. Os transdutores são os elementos primários de detecção, necessários para interagir com o mundo físico. Sensores são usados para monitorar as condições de um determinado ambiente, enquanto que os atuadores, para controlar certas entidades físicas. Na década de 1990, os conceitos relacionados com as redes de computadores foram introduzindo-se gradualmente na área de instrumentação eletrônica e, somado a este fato, a inclusão de inteligência nos transdutores fez com que fosse possível pensar em montar redes de transdutores inteligentes. Além disso, dispor de sistemas onde as atividades sejam distribuídas é hoje uma necessidade vital, impulsionada fundamentalmente pelo acentuado crescimento da Internet e das tecnologias associadas.

Atualmente, os sistemas de instrumentação distribuída envolvendo transdutores com capacidade de processamento local, constituem uma tendência na área de instrumentação. Estes sistemas são sustentados sobre a base das comunicações digitais, o emprego de sistemas abertos, ferramentas padronizadas, e tecnologias associadas à Internet. A instrumentação e o controle distribuído irão contribuir de maneira decisiva nos futuros sistemas de automação, não apenas no setor industrial, mas também em outras áreas como a biomédica, telemedicina e automação residencial.

3.2 - Sistema de Instrumentação

3.2.1 - Visão Conceitual

O fluxo de informação relacionado com os sistemas de medição e controle não envolve apenas a comunicação entre objetos e sistemas de medida, mas também a interface homem-máquina.

Nesse fluxo de informação encontram-se envolvidos três mundos diferentes, como mostrado na Figura 3.1 [37].

a) Mundo Físico: representa a magnitude medida e a variável controlada. As leis naturais regem este mundo, onde a causalidade encontra-se estritamente estabelecida.

b) Mundo Lógico: representa o sistema de processamento de informação para medição e controle. As regras lógicas regem este mundo, onde a informação é descrita através de sinais e códigos.

c) Mundo Intelectual do Ser Humano: representa o contexto interno do cérebro humano. Neste mundo a informação é traduzida em conhecimentos e conceitos.

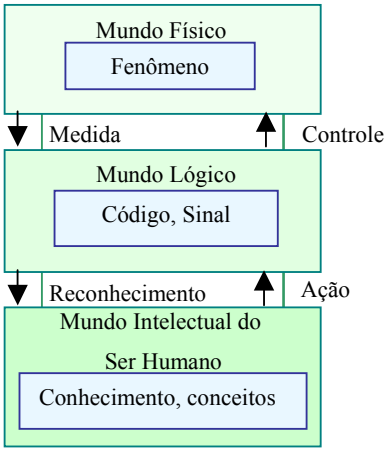


Figura 3.1. Sistema de instrumentação; uma visão conceitual.

A informação do mundo físico é extraída através de dispositivos sensores e técnicas de instrumentação e transferida para o mundo lógico. A informação é processada, então, de acordo com os objetivos do sistema de medição e controle e reconhecida, depois, pelo homem, através da interface homem-máquina, encarregada de apresentar os resultados finais. Deste modo, a informação é transferida para o mundo intelectual do ser humano para ser transformada em

conhecimento e conceitos. As intenções são logo transformadas em comandos e transmitidas ao sistema através da interface homem-máquina. Em função do comando, o sistema de controle agirá sobre uma variável física através de um dispositivo atuador. Em virtude destes conceitos, define-se "medição e controle" como o processo que envolve medição, reconhecimento, ação e controle em três diferentes mundos ou contextos, realizando-se no mundo físico, a ação desejada pelo ser humano.

3.2.2 - Dispositivo Transdutor

O termo transdutor deriva-se do latim *transducere,* que significa "conduzir através de". Um transdutor é um dispositivo capaz de converter energia de um domínio para outro, na mesma ou em diferente forma. Os sinais, assim como a energia, podem ser conduzidos através do elemento transdutor [38].

Os termos sensor e transdutor são por vezes utilizados como sinônimos. A diferença entre ambos os termos é muito sutil [39], [40]. Um sensor executa uma ação de transdução e pode ser definido como sendo um dispositivo capaz de converter uma grandeza física ou química em um sinal elétrico. Outro caso semelhante é o do atuador. Um atuador é qualquer dispositivo capaz de converter um sinal elétrico em uma grandeza física ou química e, portanto, também executa uma ação de transdução. Assim sendo, pode-se dizer que ambos os dispositivos são transdutores.

Utilizar-se-á a definição proposta pelo IEEE, definindo sensor como sendo um transdutor que converte um parâmetro físico, biológico ou químico em um sinal elétrico e, atuador como sendo um transdutor que aceita um sinal elétrico para convertê-lo em uma ação física [2], [3].

O sensor é utilizado como elemento primário para detectar as condições de um ambiente ou de um processo, por exemplo, mudanças de temperatura. O atuador, por exemplo, uma válvula ou um relé, é utilizado para executar uma ação específica com base na informação fornecida por um sensor.

Um transdutor pode ser conectado a uma rede, a fim de que as informações que fornece possam ser compartilhadas por outros dispositivos. Tipicamente, o sinal proveniente de um sensor geralmente é fraco e pode ser facilmente afetado por interferências. Com o objetivo de condicionar o sinal, existem os circuitos de condicionamento, que executam funções como amplificação, filtragem, casamento de impedância, modulação e demodulação [39]. Na seqüência, o sinal precisa ser convertido do domínio analógico para o domínio digital. Para tal

fim emprega-se um conversor analógico/digital (ADC – *Analog to Digital Converter*). O sinal em formato digital, então, pode ser enviado para uma unidade microprocessadora ou para um microcontrolador que são elementos encarregados de processar os dados. Finalmente, a informação pode ser enviada para a rede. Na Figura 3.2 é mostrada a seqüência de blocos funcionais entre o transdutor e a rede.

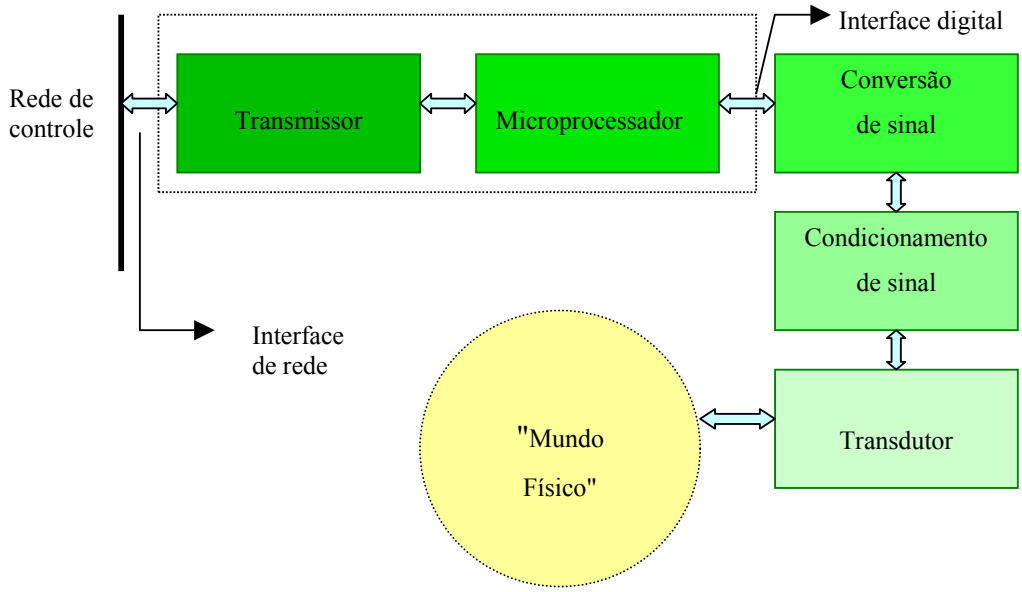


Figura 3.2. Dispositivo transdutor conectado a uma rede.

Um ponto relevante no sistema da Figura 3.2 é o intercâmbio de informação através das interfaces envolvidas. As interfaces devem ser claras e bem definidas e, deste modo simplifica-se a substituição de algum componente do sistema sem mudar a natureza das mesmas.

Um transdutor vincula-se ao nó de rede através de uma interface digital, a qual fornece uma via de comunicação entre ambos dispositivos. O mesmo acontece entre o processador de rede e a rede. O interesse principal centra-se nas interfaces de rede e na digital, embora outras interfaces estejam envolvidas no sistema. Este conceito ficará esclarecido nos capítulos sucessivos.

3.2.3 - Sensor Inteligente

Na literatura atual, existem diferentes pontos de vista sobre a definição de sensor inteligente. Em certos casos, o sensor inteligente é considerado como sendo um dispositivo que integra pelo

menos um elemento sensor e um circuito de processamento de sinal. Entretanto, a definição anterior é inconveniente porque uma ampla gama de sensores cai nessa categoria.

O termo inteligente é mais adequado para denotar a integração do dispositivo anterior com um processador, que possibilita a introdução de inteligência. A princípio, pode-se estabelecer a seguinte classificação [40]:

a) Sistema sensor: sistema não integrado composto por um sensor, circuitos de pré-processamento e um processador.

b) Sensor integrado em um sistema sensor: aquele que possui o sensor e circuitos de pré-processamento integrados e uma unidade processadora não integrada.

c) Sensor Inteligente: sistema integrado pelo sensor, os circuitos de pré-processamento e a unidade processadora.

Na Figura 3.3 ilustra-se esquematicamente a concepção básica de um sensor inteligente.

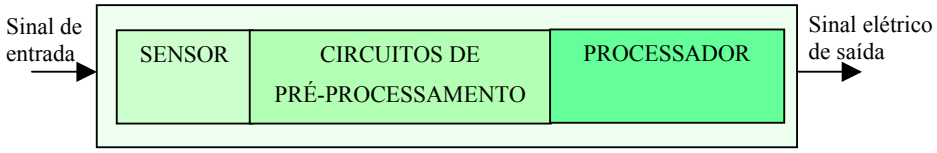


Figura 3.3. Concepção básica de um sensor inteligente.

3.2.4 - Sistemas de Medição e Controle Distribuído

Em termos simples, um sistema de Medição e Controle Distribuído (DMC – *Distributed Measurement and Control*) consiste em um conjunto de transdutores conectados através de uma rede [14]. A rede mencionada pode ser composta por diferentes níveis até atingir, por exemplo, a Internet. O objetivo de um sistema DMC é controlar um determinado processo, sendo que os transdutores são os dispositivos primários de detecção e atuação. Na Figura 3.4 é ilustrado o conceito fundamental de sistema DMC.

Geralmente, em um sistema DMC os transdutores são conectados à rede através da intermediação de nós inteligentes de rede. Nó inteligente é um módulo baseado em algum tipo de processador que: a) tem a capacidade de se identificar no sistema; b) possibilita o acesso do transdutor à rede e c) é responsável pelo processamento de dados do transdutor e da interface de

rede associada. Desta maneira, através da rede de controle, os dados associados aos transdutores podem ser disponibilizados na Internet.

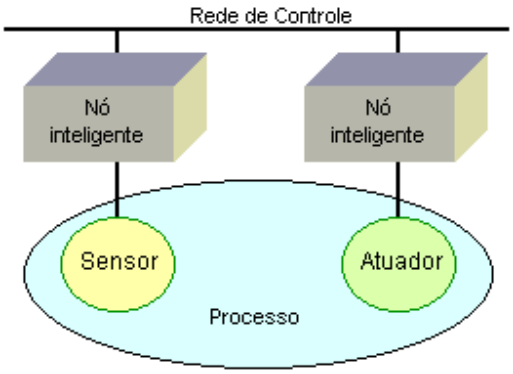


Figura 3.4. Sistema DMC elementar.

Geralmente, um nó inteligente de rede possui capacidade de processamento local, podendo tomar decisões sobre o transdutor associado. Por exemplo, quando o valor de temperatura fornecido por um sensor for maior do que um valor limite estabelecido, o nó de rede pode acionar um ventilador a fim de estabilizar o sistema. Assim, as atividades podem ser distribuídas através da rede e, ao mesmo tempo, pode se dispor de uma ou várias estações de monitoramento em um nível superior da rede. Neste contexto diz-se que o nó de rede fornece inteligência distribuída para o sistema.

Os transdutores conectados em rede possibilitam o compartilhamento das informações de um sistema de instrumentação e fazem com que seja possível supervisionar um processo completo e acessar e monitorar, de forma remota, as variáveis envolvidas.

As tendências atuais na área de instrumentação mostram as arquiteturas de Medição e Controle Distribuído (DMC) envolvendo transdutores inteligentes como sendo as candidatas a constituir a próxima geração de sistemas de instrumentação distribuída.

Com base nos conceitos até aqui estudados, a definição de transdutor inteligente reformula-se da seguinte maneira: transdutor inteligente é um dispositivo com capacidade de processamento local, habilitado para tomar decisões baseando-se no sinal de entrada, e que pode enviar ou receber dados em formato digital, facilitando as atividades em sistemas distribuídos.

A modo de esclarecimento, cita-se que um exemplo interessante é o do corpo humano composto por inúmeros elementos sensores. A tarefa mais importante de processamento de

informação do sistema de sensoriamento é extrair a informação necessária dos receptores de sinais e transmitir a informação útil para o cérebro. A informação necessária é transferida de maneira distribuída, liberando assim, a carga de trabalho do cérebro [37].

3.2.5 - Necessidade de Sistemas Distribuídos

Os sistemas distribuídos precisam de um maior "nível de inteligência" se comparados com os tradicionais sistemas centralizados, porém, são muito mais flexíveis. De um modo geral, a necessidade de se implementarem sistemas distribuídos é motivada pelos seguintes fatos:

a) Obtenção de flexibilidade, disponibilidade, escalabilidade e integração;

b) Ambientes atuais inerentemente distribuídos;

c) Informação distribuída (WWW);

d) Necessidade crescente de compartilhar informações e recursos;

e) Projeto de sistemas utilizando nós especializados, com capacidade de processamento local a fim de liberar a carga de processadores centrais.

Dentre as vantagens mais significativas dos sistemas distribuídos encontram-se a redução de custos devido à boa relação preço/desempenho, o alto rendimento devido à possibilidade de executar ações em forma paralela, o suporte de aplicações inerentemente distribuídas; por exemplo, empresas distribuídas geograficamente e a natureza aberta e heterogênea que possibilita a interoperabilidade. Entretanto, existem algumas desvantagens como os problemas de segurança e determinismo, e problemas associados às inter-redes como perdas de mensagens e saturação.

Talvez o maior desafio no projeto deste tipo de sistemas esteja relacionado com o desenvolvimento de novos tipos de *softwares*, bem mais complexos do que os tradicionais.

3.3 – Comentários Finais sobre o Capítulo 3

Os conceitos que foram apresentados serão de grande utilidade nos próximos capítulos, pois no trabalho foi desenvolvido um nó de rede padronizado contendo transdutores, para aplicações em ambientes de instrumentação distribuída. No entanto, a rede pode ser composta por vários níveis e por diferentes tecnologias para conectar transdutores em rede. Em virtude disso e devido à importância transcendental no presente trabalho, os transdutores em rede motivam o seu tratamento em um capítulo à parte, que será apresentado na seqüência.

Capítulo 4

Transdutores em Rede

Neste capítulo abordam-se as noções fundamentais relacionadas com as redes de transdutores. O capítulo começa com um breve histórico e, na seqüência, são apresentadas as diferentes técnicas de conexão de sensores em rede. Posteriormente, são abordados os tópicos referentes a redes de controle e barramentos de campo e são exploradas as tecnologias existentes no mercado atual. Finalmente são expostos os aspectos básicos relacionados com o projeto de redes de transdutores.

4.1 - Breve Histórico

As primeiras pesquisas envolvendo sensores em rede foram motivadas pelas aplicações de uso militar. Com o transcorrer dos anos, essas aplicações foram se inserindo em diversas outras áreas, impulsionadas pelas pesquisas desenvolvidas nas universidades. Hoje, as redes de sensores e atuadores são amplamente utilizadas em aplicações militares, na indústria aeroespacial, na automação industrial, em sistemas de monitoramento de ambientes e no controle de tráfego aéreo.

Outras áreas de aplicação muito atraentes são a engenharia biomédica, a automação residencial e as aplicações no campo da telemedicina, atualmente em menor escala, mas com perspectivas de grande crescimento para o futuro.

Nos finais da década de 1970, a *Advanced Research Projects Agency* (DARPA), iniciou as pesquisas na área de sensores em rede, através do programa Redes de Sensores Distribuídos (DSN - *Distributed Sensor Networks*). No ano de 1978 a DARPA patrocinou um *workshop* em que foram identificadas tecnologias para serem aplicadas em redes de sensores distribuídos como técnicas de comunicação, técnicas de processamento e algoritmos para aplicações distribuídas.

Com o transcorrer da década de 1980 várias universidades viram-se envolvidas neste tipo de pesquisa. Atualmente, as pesquisas em redes de sensores são de importância vital em diversos centros militares [41].

Realizando-se uma retrospectiva das tecnologias vinculadas às redes de sensores, podem ser citadas como exemplo, as redes de radar utilizadas para o controle de tráfego aéreo e algumas aplicações de uso militar. Durante a "Guerra fria" o *Sound Surveillance System* (SOSUS) foi empregado pelos americanos com a finalidade de detectar e rastrear submarinos soviéticos. Este sistema era composto de sensores acústicos dispostos de maneira estratégica no fundo do oceano.

Atualmente o SOSUS é utilizado por organizações oceanográficas para monitorar eventos no mar como sismos e atividades relacionadas com o mundo animal. Este último é apenas um exemplo de sistema concebido para aplicações militares, que mais tarde tornou-se uma ferramenta de pesquisa nas mãos da comunidade científica.

O desenvolvimento de redes de sensores, ou de modo mais geral, de redes de transdutores, precisa da colaboração de três diferentes áreas da engenharia: instrumentação, informática e comunicação. Portanto, pode-se afirmar que a concepção de um projeto envolvendo transdutores em rede é multidisciplinar. Este fato constitui um desafio, embora hoje existam tecnologias não disponíveis há 20 anos. Sensores, processadores e dispositivos de comunicação são hoje menores e muito mais baratos.

As perspectivas para a próxima década indicam a utilização da tecnologia embarcada, dispositivos do tamanho de partículas de pó e de peso desprezível [41].

As tecnologias disponíveis atualmente mudaram aqueles primeiros conceitos na área de sensores em rede. O uso de dispositivos sensores microeletromecânicos (MEMS), a tecnologia de rede sem fios (*wireless*), as tecnologias associadas à Internet e a grande quantidade de ferramentas para desenvolvimento de sistemas digitais são os novos protagonistas na área de instrumentação.

4.2 - Conexão de Sensores em Rede

Ainda hoje, uma maneira típica de conectar sensores com instrumentos e sistemas baseados em computador é através de configurações ponto-a-ponto ou de forma multiplexada. Estas técnicas de conexão fazem com que os sistemas de instrumentação fiquem volumosos devido à quantidade de cabos que precisam ser utilizados, além de terem custo elevado e tornarem difíceis as tarefas de manutenção. Junto à evolução acentuada das tecnologias aplicadas nas redes de computadores,

projetistas e fabricantes de transdutores tentam encontrar maneiras adequadas de adaptar essas tecnologias às redes de transdutores para serem aplicadas em instrumentação e controle.

Com o transcorrer das décadas de 1980 e 1990 diversos barramentos para sensores e redes de controle têm sido desenvolvidos para serem utilizados em diferentes aplicações, abrangendo desde o controle de processos industriais até a automação residencial. Essas tecnologias, geralmente são denominadas de barramentos para sensores (S*ensorbus*), redes de controle e redes orientadas a dispositivos (*Devicebus*) e barramentos de campo (*Fieldbus*) [42], [43].

4.2.1 - Barramentos para Sensores

O barramento para sensores ou *Sensorbus* é um sistema básico de interconexão de sensores e atuadores em sistemas de controle baseados em algum tipo de processador. Desta maneira, os dados podem ser transferidos de forma direta para um microcontrolador ou para um microcomputador que age como o controlador do sistema. A comunicação se realiza de forma bidirecional através de um barramento digital de dados e controle como mostra a Figura 4.1 [1].

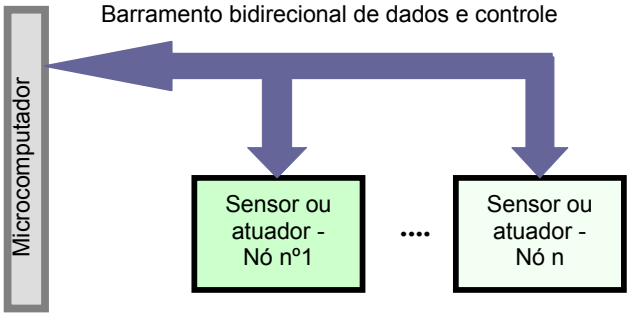


Figura 4.1. Barramento para sensores.

Cada nó conectado ao barramento pode conter um ou mais transdutores. Note-se que a transmissão de dados realiza-se sempre através do mesmo meio físico, que pode ser cabo de par trançado, cabo coaxial ou fibra óptica. A modalidade de transmissão analógica, por exemplo, a tradicional malha 4-20 mA, precisa de dois fios por variável transmitida, gerando elevados custos de instalação e manutenção [44].

Dentre os primeiros barramentos para sensores conhecidos, destaca-se o Padrão Michigan Paralelo (MPS - *Michigan Parallel Standard*), desenvolvido na Universidade de Michigan [45].

Este barramento paralelo implementa a interface entre o *host* e os nós através de 16 linhas: 8

linhas bidirecionais de dados, 4 linhas de controle para sincronizar a transferência de dados, um sinal de paridade e 3 linhas de alimentação. A configuração mencionada é apresentada na Figura 4.2.

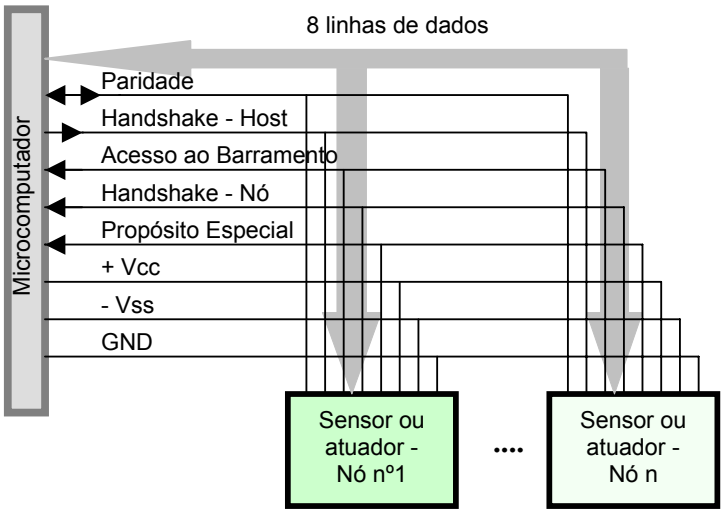


Figura 4.2. Barramento MPS.

No barramento MPS, o *host* inicia uma transferência de mensagem enviando um sinal através da linha *Handshake-Host*[11]. O primeiro *byte* enviado é um endereço que, logo depois, é decodificado por todos os nós conectados ao barramento. O nó correspondente reconhecerá o evento através de um sinal na linha *Handshake-Nó* e, a seguir, colocará um sinal em nível lógico "1" na linha Acesso ao Barramento, para impedir que os outros nós acessem o canal de comunicação. O barramento é liberado quando o sensor muda o estado do sinal de acesso. O sinal de propósito geral pode ser empregado para gerar uma interrupção e requisitar o serviço imediato por parte do *host*, ou para detectar um evento de inserção, caso um novo nó seja conectado ao barramento.

Uma outra alternativa de interfaceamento entre o *host* e os nós do sistema, denomina-se Padrão Michigan Série (MSS - *Serial Michigan Standard*), mostrado na Figura 4.3. Neste caso, a transferência de dados realiza-se de forma série, através de uma linha de dados que utiliza um

[11]*Handshake:* Termo que faz referência ao processo de estabelecimento de um contato inicial entre dois dispositivos de comunicação. A partir desse instante, ambos os dispositivos "põem-se de acordo" em utilizar um protocolo.

sinal de relógio com a finalidade de sincronizar a transmissão. Esta configuração implementa apenas 4 linhas e possui elevada imunidade ao ruído.

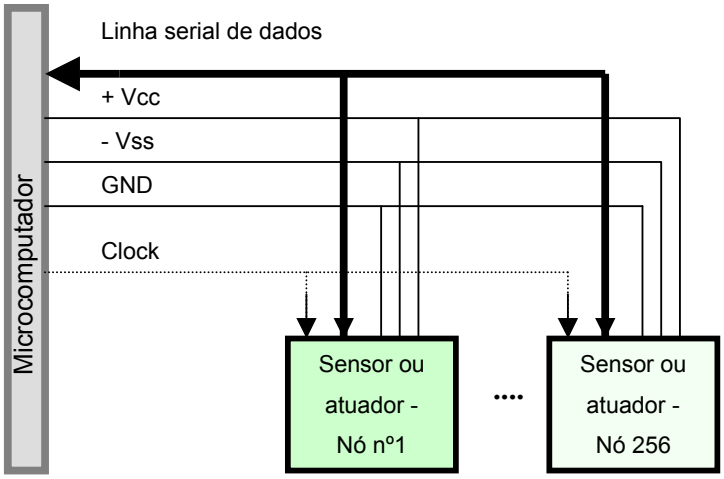


Figura 4.3. Barramento MSS.

A mensagem emitida pelo *host* contém o endereço correspondente, o tamanho da mensagem, o comando e os dados propriamente ditos, ao passo que a mensagem enviada pelo nó contém o código correspondente, o tamanho da mensagem e os dados.

Com o objetivo principal de manter os custos em um valor razoável, a Universidade de Tecnologia de Delft propôs uma configuração conhecida como *Integrated Smart Sensor* (IS^2), apresentada na Figura 4.4 [1], [46].

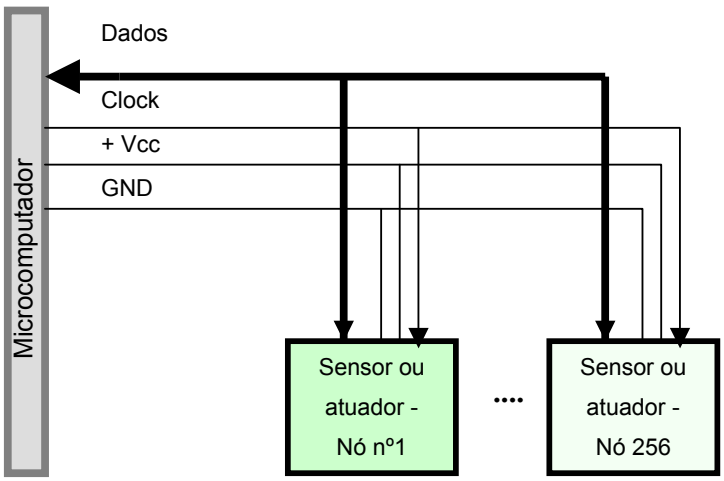


Figura 4.4. Barramento ISS.

Nesta configuração, os nós são sensores com processadores integrados que utilizam o barramento IS^2 para estabelecer a transferência de dados com um controlador central. O barramento IS^2 utiliza a técnica de comunicação mestre-escravo e é composto por 4 linhas, duas de alimentação, a linha de dados e um sinal de relógio. O mestre controla o relógio, o qual pode ser ajustado até um máximo de 400 kHz. Os dispositivos escravos podem requisitar serviço do mestre quando precisarem de atenção.

4.2.2 - Redes de Controle e Barramentos de Campo

As redes de controle e os barramentos de campo, estes últimos conhecidos comercialmente como *fieldbuses*, consistem em um barramento de comunicação digital bidirecional, que possibilita a interconexão de nós de rede. Cada nó pode conter um ou mais transdutores com algum grau de capacidade computacional localizada. Um *host* da rede pode se comunicar com estes dispositivos utilizando um determinado protocolo para implementar ações de monitoramento e atuação. Na Figura 4.5, ilustra-se de maneira simplificada, a utilização de um barramento de campo para conectar transdutores com uma estação de operação e um microcomputador em que pode rodar uma ferramenta para manutenção.

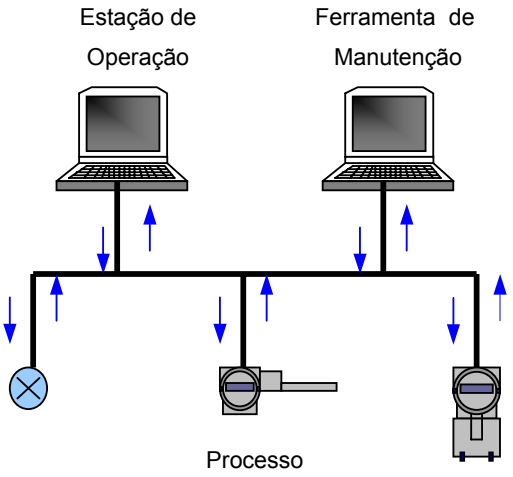


Figura 4.5. Barramento de campo.

As redes de controle são barramentos de alto nível que executam algum tipo de processamento de dados em cada nó de rede antes de serem passados para o *host* correspondente.

Já os barramentos de campo incorporam a camada de usuário alocada acima da camada de aplicação do modelo OSI e podem executar procedimentos de controle tanto no dispositivo de campo como no controlador. A tecnologia *fieldbus* diferencia-se também pelo fato de introduzir segurança intrínseca para dispositivos de campo, em ambientes perigosos ou de risco [43]. Neste caso, as instalações são alimentadas através de barreiras. Ou seja, circuitos que limitam os valores de energia presentes em um dispositivo, com a finalidade de evitar acidentes em ambientes com perigo de explosão.

No projeto de um sistema envolvendo barramentos de campo devem ser avaliadas algumas características fundamentais para que o sistema implementado possa funcionar da maneira desejada. Dentre estas características destacam-se as seguintes:

a) Camadas implementadas em relação ao modelo OSI;
b) Topologia;
c) Método de comunicação utilizado;
d) Quantidade de nós possíveis a serem conectados;
e) Algoritmo de acesso ao meio;
f) Velocidade de transmissão de dados;
g) Meio físico de transmissão;
h) Opções de conectividade;

a) Camadas em Relação ao Modelo OSI

Os barramentos de campo implementam apenas algumas camadas pertencentes ao modelo de referência ISO/OSI. Geralmente, define-se a camada física, a de enlace de dados e a de aplicação subdividida em outras camadas. Desta maneira, o modelo fica dividido em nível físico, nível de comunicação e nível de usuário [43].

b) Topologia

Em um projeto envolvendo barramentos de campo são possíveis várias topologias. Dentre estas configurações destacam-se a topologia em barramento com derivações, a ponto-a-ponto, em árvore e a topologia *end to end*. Na topologia em barramento com derivações, utiliza-se um barramento através do qual se conectam diferentes equipamentos ou barramentos secundários (*spurs*), sendo que cada *spur* pode conter diferentes dispositivos. Na topologia ponto-a-ponto

implementa-se a conexão serial de dispositivos. Neste caso, a instalação deve conter conectores, para que a desconexão de um determinado dispositivo não venha interromper a continuidade da linha. A topologia em árvore enlaça vários dispositivos através de acopladores, que por sua vez, se conectam ao barramento principal. A topologia *end to end* emprega-se no caso de serem conectados dois equipamentos diretamente. As diferentes topologias são apresentadas na Figura 4.6.

Finalmente, cabe destacar que é possível, também, implementar a topologia mista. Nesse caso, as topologias em barramento com derivações, em árvore e ponto-a-ponto encontram-se vinculadas.

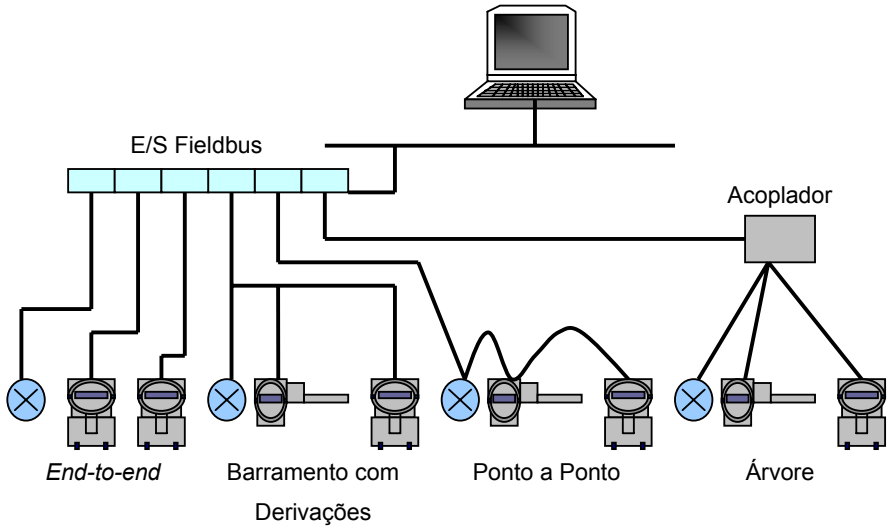


Figura 4.6. Diferentes topologias de Barramentos de campo.

c) Métodos de Comunicação

O método de comunicação faz referência à interação entre os nós que compõem o sistema, tendo-se interações distribuídas, centralizadas e híbridas. No método produtor-consumidor, muito comum em barramentos de campo, o nó produtor coloca as variáveis em *buffers*[12], sendo que qualquer outro nó da rede pode acessar esses dados. Esta técnica de comunicação é a mais eficiente para a transferência de dados entre vários usuários. O controlador consome a variável produzida por um sensor determinado e produz a saída utilizada pelo atuador associado [43].

[12] *Buffer*: Elemento intermediário para armazenamento temporário.

Outra das técnicas típicas utilizadas em barramentos de campo é a mestre-escravo. Nessa técnica, o dispositivo mestre recebe a informação emitida pelo nó escravo, faz o processamento, toma decisões e pode redistribuir as informações para os outros nós do sistema. Esta técnica utiliza o modelo centralizado, enquanto que o método produtor-consumidor permite a implementação de modelos distribuídos.

Outro método de comunicação em ambientes de rede é o *peer to peer* ou igual-a-igual, em que a comunicação entre dois nós se realiza sem passar por um servidor centralizado.

d) Quantidade de Nós

A quantidade de nós a serem implementados no sistema depende da tecnologia utilizada. Os nós podem ser sensores, atuadores, CLPs e interfaces homem-máquina.

e) Algoritmo de Acesso ao Meio

O nível de enlace de dados do modelo ISO/OSI garante que os dados cheguem aos pontos corretos. Dentro da camada de enlace de dados existe uma subcamada denominada de Controle de Acesso ao Meio (MAC – *Medium Access Control*). A subcamada MAC tem a função de descartar os *bits* correspondentes ao preâmbulo e ao delimitador do quadro de dados e verificar a validade do endereço de destino. A seguir o quadro passa para outra subcamada de controle, denominada de Controle de Enlace Lógico (LLC – *Logical Link Control*).

A passagem de *token* (*token passing*), é uma das formas mais comuns de acessar a rede. O método baseia-se na circulação de um sinal de um nó para outro, até atingir-se um nó que requisite uma ação de transmissão. O método pode implementar um anel lógico (*token ring*) ou um barramento (*token bus*). No método de comunicação mestre-escravo, existe também o algoritmo de resposta imediata, em que o mestre habilita uma estação a fim de que possa responder com uma mensagem. Outro algoritmo de acesso ao meio, comum nas tecnologias de rede local, é o CSMA, que será abordado no Capítulo 6.

f) Velocidade de Transmissão

A velocidade de transmissão de dados varia de acordo com a tecnologia aplicada, dependendo fundamentalmente das distâncias de cabeamento implementadas. A velocidade de transmissão

para barramentos de campo se expressa em Kbps ou Mbps, como acontece com as redes de computadores.

g) Meio Físico de Transmissão

A escolha do meio de transmissão mais conveniente para cada aplicação, precisa de uma avaliação minuciosa de aspectos tais como atenuação do sinal transmitido, influência de interferências do ambiente, comprimento máximo de cada segmento implementado, flexibilidade e custos. O cabo de par trançado é susceptível de atenuação no sinal transmitido, mas em compensação introduz flexibilidade de instalação e possui custo muito baixo. A fibra óptica admite grandes distâncias de cabeamento, entretanto, o custo de implementação é elevado.

h) Opções de Conectividade

A conectividade refere-se à integração de equipamentos e dispositivos através de tecnologias de rede. O conceito de conectividade está relacionado com a utilização de tecnologias abertas e padronizadas. Levando em consideração este fato, ao projetar um sistema de instrumentação envolvendo tecnologias de rede, devem-se avaliar quais as opções de conectividade a fim de que o sistema se torne flexível para mudanças futuras.

Embora a implementação de um barramento de campo possa exigir o conhecimento de conceitos complexos por parte do projetista do sistema, sua correta implementação introduz vantagens significativas, dentre as quais, podem se mencionar:

a) Redução de cabeamento: as variáveis de processo como temperatura, pressão, vazão, etc; se transmitidas de forma analógica, precisam de um par de fios por cada variável transmitida. Com a implementação de um barramento de campo, as informações são transmitidas através do mesmo meio físico, por exemplo, um cabo de par trançado.

b) Facilidade de se realizarem mudanças futuras: o barramento de campo torna o sistema mais flexível, facilitando a escalabilidade.

c) Facilidade de execução de tarefas de calibração e manutenção: os dispositivos conectados ao barramento trazem hoje funções que executam configuração e diagnóstico.

d) Modularidade: os sistemas de instrumentação tornam-se modulares, possibilitando acrescentar ou retirar dispositivos da rede com relativa facilidade.

4.2.3 - Redes e Barramentos de Campo Comerciais

No mercado atual, existem diversas tecnologias factíveis de serem aplicadas em redes de transdutores, porém, nota-se a falta de um padrão internacional de interfaceamento para este tipo de aplicações. Dentro desta grande diversificação de tecnologias, existem soluções de domínio aberto, outras padronizadas parcialmente e uma ampla quantidade de soluções proprietárias.

A seguir, apresenta-se uma lista detalhando as características das principais tecnologias utilizadas na atualidade, abrangendo-se protocolos, barramentos para sensores, barramentos de campo e redes de controle. Salienta-se que, neste tópico, são apresentadas apenas algumas das alternativas possíveis.

a) ASI (*Actuator Sensor Interface*)[13] é um sistema de interconexão de transdutores de baixo custo, desenvolvido na Alemanha por um consórcio de fornecedores de transdutores e projetado especificamente para operar em ambientes industriais, em nível de processo. Esta tecnologia funciona com 4 *bits* por mensagem, implementando o método de comunicação mestre-escravo. Pode operar com distâncias de até 100 m e 31 nós por rede, gerenciados por um controlador mestre [47].

b) CAN (*Controller Area Network*) é um barramento serial inicialmente concebido para montagem de redes internas em veículos. Foi idealizado na Alemanha por *Robert Bosch* nos começos da década de 1980. Posteriormente a *Philips* e a *Intel* introduziram o controlador do protocolo CAN. Atualmente, esta tecnologia é utilizada em diversas outras áreas da indústria. Em geral, CAN pode ser utilizado em ambientes onde for preciso comunicar vários dispositivos ou sistemas baseados em microcontrolador sob o método de comunicação *peer-to-peer*. O protocolo CAN implementa as camadas 1 e 2 do modelo OSI. Foi introduzido oficialmente em 1986 e padronizado através do *standard* ISO 11898.

Em referência à camada de enlace de dados do modelo OSI, no protocolo CAN, o algoritmo de acesso ao meio é baseado no mecanismo *Carrier Sense Multiple Access* (CSMA), comum nas redes locais [48]. Quanto ao meio físico, CAN pode utilizar cabo de par trançado, fibra óptica,

[13] Neste trabalho, a citação de tecnologias comerciais tem fins didáticos e não representa qualquer recomendação ou indicação de um fabricante.

cabo coaxial e RF (Rádio Freqüência) como meio de transmissão, suportando velocidades de até 1 Mbps. No ano de 1992 foi formado, na Alemanha, o grupo *CAN-in-Automation* (CIA) composto por fabricantes e usuários com a finalidade de divulgar a tecnologia CAN aplicada em automação [49].

c) CEbus (*Consumer Electronic Bus*) é uma tecnologia de rede utilizada principalmente para automação residencial, desenvolvida por um consórcio de fabricantes de equipamentos eletrônicos. Suporta velocidades de até 10 Kbps, podendo utilizar cabo coaxial, RF, infravermelho (IR) e a rede de energia elétrica como meio de transmissão. Atualmente o consórcio mencionado está introduzindo o conceito "*Home Plug & Play*", ou seja, a possibilidade de conectar e operar dispositivos eletrônicos de diferentes fornecedores em ambientes de automação residencial [50].

d) DeviceNet é um barramento de campo introduzido pela empresa *Allen Bradlley* para utilização em automação de processos industriais. Implementa as camadas 1, 2 e 7 do modelo OSI e foi concebido sobre a base do protocolo CAN. Utiliza um cabo de 4 fios (2 de sinal e 2 de alimentação) como meio físico de transmissão. A tecnologia *DeviceNet* é capaz de suportar 64 nós com velocidades de 500 Kbps para distâncias de até 100 m e 125 Kbps a 500 m. Devido a sua notável penetração no mercado, motivou-se a formação do grupo *Open DeviceNet Vendors Association* (ODVA) [51].

e) Ethernet é um padrão de redes LAN, cuja eficiência está totalmente comprovada em redes administrativas de baixo custo. As velocidades atuais fazem dela uma alternativa muito interessante para ser empregada em nível de campo. A tecnologia *Ethernet* será abordada em um tópico específico desta tese.

f) Foundation Fieldbus é um barramento de campo que surgiu como resultado da união das especificações de componentes em conformidade com fabricantes suportes da *Factory Information Protocol (WorldFIP)* e a associação Projeto de Sistemas Interoperáveis (ISP - *Interoperable Systems Project*). A organização *Fieldbus Foundation* foi criada com o objetivo de testar e provar componentes de barramentos de campo para criar um barramento universal.

Levando em conta este fato, *Foundation Fieldbus* não deve ser considerado apenas como sendo um barramento de campo digital, mas como um sistema de controle completo para automação industrial [52].

g) Hart (*Highway Addressable Remote Transducer*) é um protocolo de comunicação industrial produzido pela empresa *Rosemount* e utilizado em sistemas de instrumentação. O protocolo *HART* utiliza uma técnica digital de modulação de freqüência, sendo que o valor 2200 Hz corresponde a um "0" lógico e 1200 Hz corresponde a um "1" lógico. Esta técnica é utilizada para sobrepor uma comunicação digital em uma malha de corrente 4-20 mA, conectando o sistema central a um transmissor no campo. No começo da década de 1990 foi formada a *HART Communication Foudation,* com a finalidade de divulgar produtos, eventos e tecnologias associados com *HART* [53].

h) Interbus é um barramento de campo desenvolvido pela empresa *Phoenix Contact* para aplicações em controle de processos, caracterizando-se por ter elevado grau de determinismo. Nesta tecnologia é utilizada topologia em anel, com velocidade de transmissão máxima de 500 Kbps, e modelo de interação tipo mestre-escravo. Uma rede montada com tecnologia Interbus pode conter até 4096 entradas e saídas digitais [54].

i) LonWorks (*Local Operating Networks*) é uma rede de controle orientada a dispositivos introduzida pela empresa *Echelon* para aplicação em automação industrial, residencial e outras áreas. Em termos gerais, possui semelhança com as tecnologias de redes de área local. Suporta cabo de par trançado, coaxial, fibra óptica, RF e IR com velocidades de transmissão de até 1,25 Mbps para distâncias até 500 m e 78 Kbps para distâncias de aproximadamente 2000 m [55].

j) Profibus (*Process Field Bus*) é uma tecnologia que pode ser utilizada em diferentes níveis de um processo industrial. Suportada pela empresa *Siemens*, *Profibus* originou-se na Alemanha e foi padronizada através do DIN 19245. Ela é composta por 4 partes, sendo que a primeira e a segunda parte, conhecidas como *Profibus-FMS* (*Fieldbus Message Specification*), consistem em uma solução de propósito geral para atividades de comunicação em nível de controle. A terceira parte, conhecida como *Profibus-DP (Distributed Peripherals)* é um sistema de alta velocidade

para automação industrial em nível de campo. Finalmente, a quarta parte, *Profibus-PA (Process Automation)* abrange a área de controle de processos [56], [57].

k) SDS (*Smart Distributed System*) é um sistema criado pela *Honeywell-Microswitch* para conectar dispositivos inteligentes em ambientes distribuídos de automação industrial. Implementa as camadas 1, 2 e 7 do modelo OSI e foi concebido sobre a base do protocolo CAN. Utiliza um cabo de 4 fios (2 de sinal e 2 de alimentação) como meio físico de transmissão. Pode suportar até 128 nós com velocidades de até 1,25 Mbps para conectar CLPs, microcomputadores e outros dispositivos [1], [58]

l) Sercos (*Serial Real-Time Communication System*) é um barramento para comunicação serial desenvolvido na Europa para aplicações gerais em controle de motores e caracteriza-se por possuir um elevado grau de determinismo. Nesta tecnologia, padronizada através do IEC 61491, a troca de dados entre o controlador e os periféricos se dá através de anéis de fibra óptica, proporcionando assim, imunidade a interferências eletromagnéticas. Cada anel pode conter até 254 nós escravos e um 1 nó mestre. A tecnologia SERCOS motivou a formação de uma organização denominada *Interest Group SERCOS-Interface* (IGS) [59].

m) WorldFIP (*Factory Information Protocol*) é um barramento de campo idealizado na França, na década de 80, que posteriormente estendeu-se para outros países europeus. É utilizado em controle de processos e permite várias topologias de implementação. Foi padronizado através de IEC 61158 e EN 50170 [60]. Esta tecnologia baseia-se nas camadas 1, 2 e 7 do modelo OSI. As velocidades de transmissão vão desde 31,25 kbps até 2,5 Mbps, podendo utilizar par trançado, fibra óptica e RF como meio de transmissão.

4.3 - Aspectos Básicos Relacionados com o Projeto de Redes de Transdutores

De um modo geral, o projeto de sistemas distribuídos envolvendo transdutores consiste na escolha dos nós de rede e suas capacidades inerentes, a coordenação das ações e comunicações entre os mesmos e o gerenciamento de qualquer dado resultante associado. Portanto, devem ser analisadas as propriedades relacionadas com os transdutores, com os nós de rede e as propriedades relacionadas com o sistema [19].

4.3.1 - Propriedades Associadas aos Transdutores

- ✓ Variável física a ser medida: pressão, temperatura, etc;
- ✓ Características de entrada e saída do transdutor: variação de tensão, mudança no valor da resistência, sinal digital, etc;
- ✓ Desempenho: exatidão, precisão, resolução, largura de banda, etc;
- ✓ Calibração;
- ✓ Identidade: Descrição do tipo de transdutor, número de série, etc;
- ✓ Limites de utilização: valores máximos e mínimos, estabilidade, etc.

4.3.2 - Propriedades Relacionadas com os Nós de Rede e o Sistema

- ✓ Identidade: Identificação no sistema;
- ✓ Localização;
- ✓ Métodos de comunicação entre os nós: cliente/servidor, *peer to peer*, etc;
- ✓ Tecnologia de rede a ser empregada;
- ✓ Gerenciamento das propriedades de transporte de informação e comunicações entre os nós: controle de fluxo, segurança de entrega, etc;
- ✓ Sincronização;
- ✓ Determinismo[14];
- ✓ Modelos de distribuição;
- ✓ Modelagem de *software*.

Em um sistema distribuído, as propriedades relacionadas com os transdutores são gerenciadas pelo nó de rede que contém o transdutor. Portanto, para administrar de modo eficiente as informações em nível de nó de rede faz-se necessário um nível de inteligência distribuída no sistema. Desta maneira, cada nó deve possuir capacidade de processamento local. É o *software* do sistema e subsistemas de instrumentação o ponto relevante, pois, ele introduz um alto grau de integração e transparência, que são fundamentais em um ambiente distribuído.

A fim de resumir as idéias expostas, na Tabela 4.1 são apresentados os possíveis atributos de uma rede de transdutores.

[14] Determinismo: refere-se à capacidade de um sistema para determinar com alta probabilidade, o tempo que demanda uma tarefa para ser executada. Quando o sistema possui essa capacidade é chamado de determinístico.

Tabela 4.1. Possíveis atributos de uma rede de transdutores.

Atributo	Descrição	
Transdutor	Tamanho	- Reduzido ex.: MEMS, microsensores - Grande ex.: Radares
	Número	- Reduzido - Elevado
	Tipo	- Sensor, atuador
	Composição	- Homogênea (transdutores do mesmo tipo) - Heterogênea (transdutores de diferentes tipos)
	Distribuição / dinâmica	- Distribuídos e fixos ex.: sensores em asas de avião - Distribuídos e móveis ex.: sensores em veículos robô
Variável física associada	Abrangência	- Distribuída - Localizada
	Mobilidade	- Estática - Dinâmica
	Natureza	- Cooperativa ex.: Controle de tráfego aéreo - Não cooperativa ex.: Alvos militares
Ambiente de operação	Favorável	- Indústria - Automação residencial
	Adverso	- Aplicações militares em campo de batalha
Comunicação	Conexão em rede	- Através de fios - Sem fios
	Largura de banda	- Reduzida - Ampla
Arquitetura de processamento	Modelo	- Centralizado - Distribuído - Híbrido
Tecnologia de Rede	Tipo	- Barramento para sensores ex.: ISS - Barramento de campo ex.: DeviceNet - Rede de controle ex.: LonWorks - Tecnologia LAN ex.: Ethernet - Padrão de interfaceamento ex.: IEEE 1451

Fonte: Chong et al, Sensor Networks: Evolution, Opportunities and Challenges. Adaptado.

4.4 - Comentários Finais sobre o Capítulo 4

Como analisado neste Capítulo, a escolha de uma tecnologia de rede para uma aplicação específica envolvendo transdutores pode se tornar um dilema. Embora esforços na área de padronização tenham sido realizados ao longo dos anos, foi difícil estabelecer um consenso geral.

O padrão de interfaceamento IEEE 1451 emerge com uma solução criteriosa para conectar transdutores inteligentes em nível de campo, de uma maneira padronizada, flexível e útil para integrar diferentes tecnologias de rede. Visando sua aplicação, o padrão IEEE 1451 será apresentado e analisado detalhadamente no próximo capítulo.

Capítulo 5

Padrão de Interfaceamento IEEE 1451

No presente capítulo são abordados os principais aspectos do padrão de interfaceamento IEEE 1451, para conectar transdutores inteligentes em ambientes de rede. Faz-se ênfase nos padrões IEEE 1451.1 e IEEE 1451.2, cujas especificações foram a base das implementações realizadas neste trabalho.

5.1 - Introdução

Na segunda metade da década de 1980 começaram a ser empregados os microprocessadores nas tecnologias de redes industriais. A chamada segunda geração de redes industriais vislumbrava já um fato que viria a ser predominante na década de 1990: o crescimento acentuado da quantidade de tecnologias associadas aos barramentos de campo.

Se bem que, o fato anterior acarretou um notável desenvolvimento tecnológico nesta área, tornou-se cada vez mais difícil implementar as interfaces entre transdutores e diferentes tipos de rede, em virtude da variedade de opções disponíveis, porém, muitas delas proprietárias. Foi assim que começaram a surgir as seguintes questões:

a) Como fazer com que uma determinada aplicação envolvendo transdutores suporte diversas tecnologias de rede?

b) Qual a política a ser adotada pelos fabricantes de transdutores: fornecer produtos proprietários que possam funcionar apenas com uma determinada plataforma ou tentar conceber produtos flexíveis o suficiente para se adaptarem às diferentes tecnologias?

O dilema veio ficar um pouco mais esclarecido com o advento das primeiras idéias de padronização na área de interfaceamento de transdutores em ambientes de rede. Desta maneira, o paradigma de interfaceamento mudou o seu ponto vista. Isto é, ao invés de fabricar transdutores que suportem diversas tecnologias e protocolos de comunicação, resulta muito mais conveniente padronizar a interface entre transdutores e nós de rede, e entre nós e protocolos, fazendo com que

a escolha de transdutores seja independente da escolha do tipo de rede. Neste fato cimenta-se o padrão de interfaceamento IEEE 1451 para transdutores inteligentes.

O padrão de interfaceamento IEEE 1451 é resultado do esforço iniciado pelo Comitê Técnico em Tecnologia de Sensores (TC-9 - *Committee on Sensor Technology*) da Sociedade de Instrumentação e Medidas do IEEE (IEEE *Instrumentation and Measurement Society*) e pelo Instituto Americano de Padrões e Tecnologia (NIST - *National Institute of Standards and Technology*). O TC-9 junto ao NIST iniciaram, na década de 1990, uma série de *workshops* sobre padronização de interfaces que deram origem aos grupos de trabalho para criarem as especificações pertencentes à família 1451 [5], [61].

5.2 - Objetivos

O padrão IEEE 1451 define um conjunto comum de interfaces para conectar transdutores com sistemas baseados em microprocessador, instrumentos e barramentos de campo na forma "rede-independente" [17].

As características e objetivos fundamentais apresentam-se a seguir [62]:

a) O padrão é um conjunto de especificações que tem como objetivo simplificar a conectividade de transdutores;

b) Foi desenvolvido como um conjunto de especificações de domínio aberto de acordo com o consenso da indústria;

c) Seus propósitos são:

- Fornecer um modelo geral de um dispositivo transdutor, quanto a dados, controle, configuração e calibração;
- Fornecer um conjunto comum de interfaces para conectar transdutores com redes de controle e de campo.

O IEEE 1451 não é uma outra rede de campo, mas um padrão aberto que pode ser utilizado com diferentes redes. A família IEEE 1451 divide-se em seis componentes. Atualmente têm-se quatro padrões aprovados (IEEE 1451.1, IEEE 1451.2, IEEE 1451.3 e IEEE 1451.4) e três especificações em desenvolvimento (IEEE P1451.0, IEEE P1451.5 e IEEE P1451.6) [10]. No apêndice B há uma explicação mais detalhada de cada uma das diretrizes.

5.3 - Efeito do Padrão sobre um Transdutor Conectado a uma Rede

No contexto das especificações estabelecidas pelo padrão IEEE 1451, um dispositivo transdutor é considerado inteligente, se for capaz de implementar a funcionalidade necessária para simplificar a integração do transdutor em um ambiente de rede. Isto abrange desde a identificação do transdutor na rede até o tratamento das informações associadas. Na Figura 5.1, mostra-se um transdutor inteligente conectado a uma rede na forma convencional. Na Figura 5.2, ilustra-se a divisão básica entre módulos proposta pelo padrão IEEE 1451.

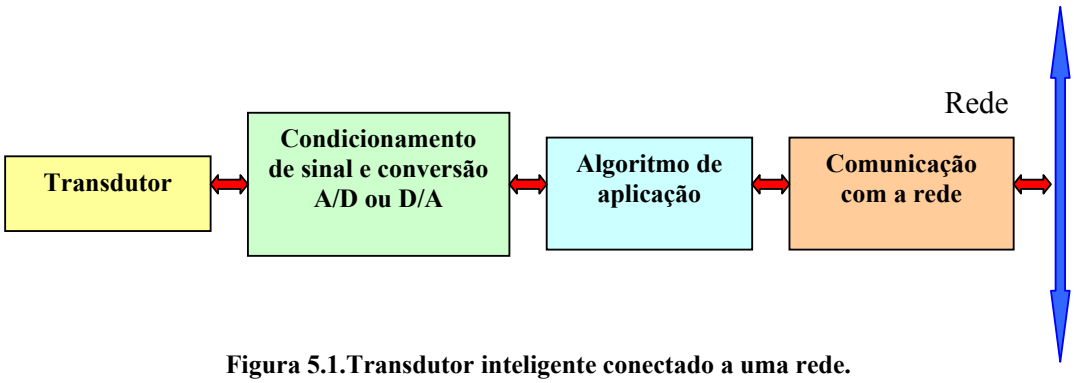


Figura 5.1.Transdutor inteligente conectado a uma rede.

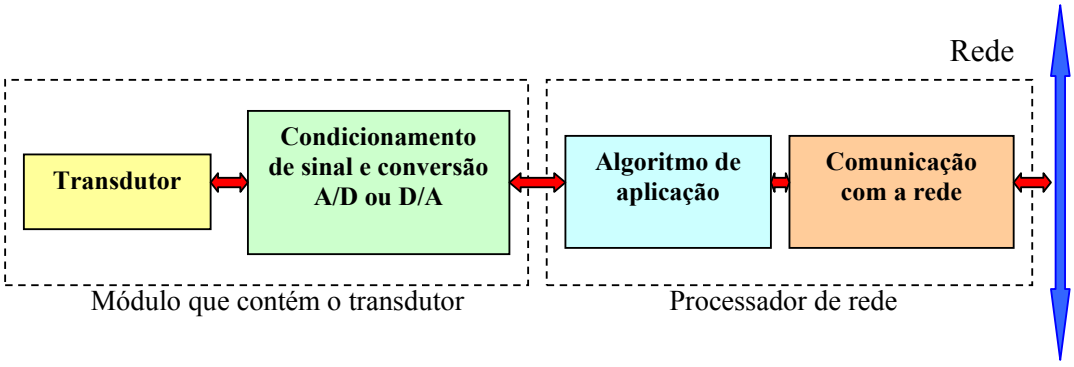


Figura 5.2. Proposta IEEE 1451.

Como ilustrado na Figura 5.2, o padrão estabelece uma divisão do sistema em dois módulos, um deles contendo o transdutor e o outro contendo os elementos necessários à comunicação do transdutor com a rede. Esta divisão da origem ao padrão IEEE 1451.2 e ao padrão IEEE 1451.1.

O padrão IEEE 1451.1 introduz o conceito de Processador de Aplicação com Capacidade de Operar em Rede (NCAP – *Network Capable Application Processor*). O NCAP é um módulo que contém tanto elementos de *hardware* quanto de *software*, salientando-se que, como parte dos

elementos de *software*, o NCAP contém a descrição de transdutores através de um modelo orientado a objeto, introduzindo um elevado grau de abstração. O padrão IEEE 1451.2 introduz o conceito de Módulo de Interface para Transdutores Inteligentes (STIM – *Smart Transducer Interface Module*). O STIM contém os elementos necessários de um sistema de aquisição de dados e um dispositivo de memória não volátil, onde são armazenadas as informações dos transdutores conectados, em formato eletrônico, dando origem ao bloco Informações de Transdutores em Formato Eletrônico (TEDS – *Transducer Electronic Data Sheet*). Este padrão estabelece, além do protocolo de comunicação necessário, uma interface padronizada entre o STIM e o NCAP, denominada Interface Independente de Transdutores (TII – *Transducer Independent Interface*). O conceito é ilustrado na Figura 5.3.

Cumpre salientar que no presente trabalho são de particular interesse o padrão IEEE 1451.1 e IEEE 1451.2.

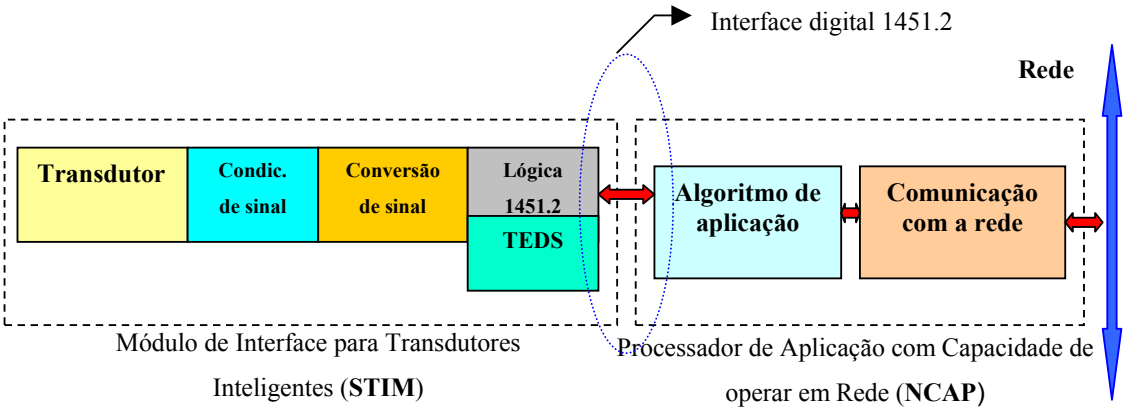


Figura 5.3. Arquitetura da interface IEEE 1451.

5.4 - O Padrão IEEE 1451.2

Cronologicamente, o padrão IEEE 1451.2 foi a primeira diretriz aprovada da família IEEE 1451. Portanto, primeiramente será analisado o padrão IEEE 1451.2 e na seqüência será abordado o padrão IEEE 1451.1. A norma IEEE 1451.2 é mais fácil de ser entendida e, além disso, a especificação está associada com a implementação do primeiro módulo na seqüência que abrange

desde o nível de entradas/saídas até o receptor final das informações na rede; em função disso, e como já foi dito, o padrão IEEE 1451.2 será abordado em primeiro lugar.

O padrão IEEE 1451.2 faz parte da família IEEE 1451 e seus principais objetivos são os seguintes [3]:

a) Fornecer capacidade *plug and play* para conectar transdutores em ambientes de rede;

b) Possibilitar e simplificar a criação de redes de transdutores inteligentes;

c) Facilitar o suporte de vários fornecedores.

Em simples termos, o padrão IEEE 1451.2 especifica a capacidade *plug and play* (conecte e opere) de um dispositivo transdutor. O termo *plug and play* tem sua origem no âmbito da ciência da computação e faz referência à possibilidade de evitar as dificuldades de reconfigurar um sistema, toda vez que é acrescentado um novo dispositivo nele. O conceito anterior é levado pelas especificações do padrão IEEE 1451.2, ao âmbito das redes de transdutores inteligentes. Isto é possível, desde que as informações gravadas nas TEDS façam parte do transdutor. Assim, um determinado STIM contendo transdutores pode ser retirado de um determinado NCAP e ser conectado a outro, num outro ponto da rede, sem mudar a natureza da interface.

Cumpre salientar que, embora os circuitos de pré-processamento e os transdutores propriamente ditos façam parte do STIM, no padrão IEEE 1451.2 não se pretende especificar os aspectos relacionados com o condicionamento e conversão de sinal, nem como as TEDS são utilizadas em uma determinada aplicação. A finalidade do padrão é fornecer um conjunto mínimo de ferramentas que possibilitem a identificação de transdutores, além de fazer com que os sistemas implementados sejam escaláveis [63].

5.4.1 - Módulo de Interface para Transdutores Inteligentes (STIM)

O módulo STIM pode ser composto por um ou vários transdutores, circuitos de condicionamento de sinal, conversores A/D e D/A necessários para realizar o interfaceamento dos transdutores com o processador[15] local, um dispositivo de memória não volátil para armazenar os formatos TEDS, acessível através do processador local, e a lógica necessária para implementar a TII.

É importante notar que, no contexto do padrão, os transdutores consideram-se como sendo parte do STIM. De fato, para fornecer a auto-identificação de cada transdutor, eles devem

[15] Fala-se em processador de um modo geral. O processador pode ser um microcontrolador, um dispositivo lógico programável ou um circuito integrado de aplicação específica.

permanecer inseparáveis do STIM durante seu funcionamento normal [64],[65]. Nesse contexto, cada transdutor que faz parte do STIM é chamado de canal transdutor. Cada canal é considerado inteligente devido aos três fatos mencionados a seguir [63]:

a) É descrito através das especificações armazenadas, em formato eletrônico, em um dispositivo de memória não volátil (TEDS);

b) O controle e os dados associados são digitais;

c) São fornecidas funções de controle, estado e disparo para suportar a funcionalidade própria de cada canal.

O STIM é controlado pelo módulo NCAP via interface TII. Deste modo, quando conectado a uma rede através do NCAP, o conjunto é "visto" pela rede como um ponto de inteligência distribuída. Este fato é fundamental, pois possibilita o funcionamento de transdutores inteligentes em sistemas de medição e controle distribuído.

Apenas a interface TII entre o STIM e o NCAP é padronizada pelo IEEE. As outras interfaces, por exemplo, entre o transdutor e os circuitos de condicionamento ou entre os circuitos de condicionamento e o conversor, ficam ao critério do projetista. Ou seja, pode ser usado qualquer tipo de conversor e circuito de condicionamento, sendo necessário que as TEDS estejam corretamente implementadas de acordo com as informações do transdutor. Deste modo, através da implementação livre dessas interfaces, pretende-se continuar introduzindo a necessária diferenciação de produtos no que diz respeito ao desempenho, qualidade, apresentação e custo [17].

O bloco correspondente à lógica 1451.2, mostrado no STIM da Figura 5.3, pode ser implementado de diferentes maneiras, por exemplo, com microcontrolador de baixo custo, PLD ou ASIC.

A implementação do módulo STIM é bastante flexível e sua concepção pode adotar diversas formas. Visando esclarecer essa idéia, na Figura 5.4 são apresentadas algumas possibilidades de implementação com microcontrolador [65].

Na Figura 5.4a mostra-se um STIM com um canal transdutor, no caso, um simples sensor analógico. Esta aplicação é denominada de STIM dedicado e pode ser utilizada quando for preciso implementar um módulo específico para detectar as condições de um determinado ambiente através de um sensor. No exemplo da Figura 5.4b, apresenta-se um módulo STIM com entradas e saídas digitais.

Os formatos TEDS, neste caso, permitem realizar a descrição de um STIM de 8 canais, ou em forma alternativa, de um STIM de dois canais, um de entrada e outro de saída, cada um deles composto por quatro canais transdutores. No caso da Figura 5.4 c, mostra-se um módulo STIM com diversos sensores analógicos. Essa configuração pode ser empregada, por exemplo, para detectar as condições de um líquido que faz parte de um determinado processo. Finalmente, no último exemplo, ilustra-se um STIM com sensores e atuadores. Esse caso é conveniente quando for preciso que um atuador execute alguma ação baseada em algum algoritmo de controle. O algoritmo pode rodar no STIM ou no NCAP.

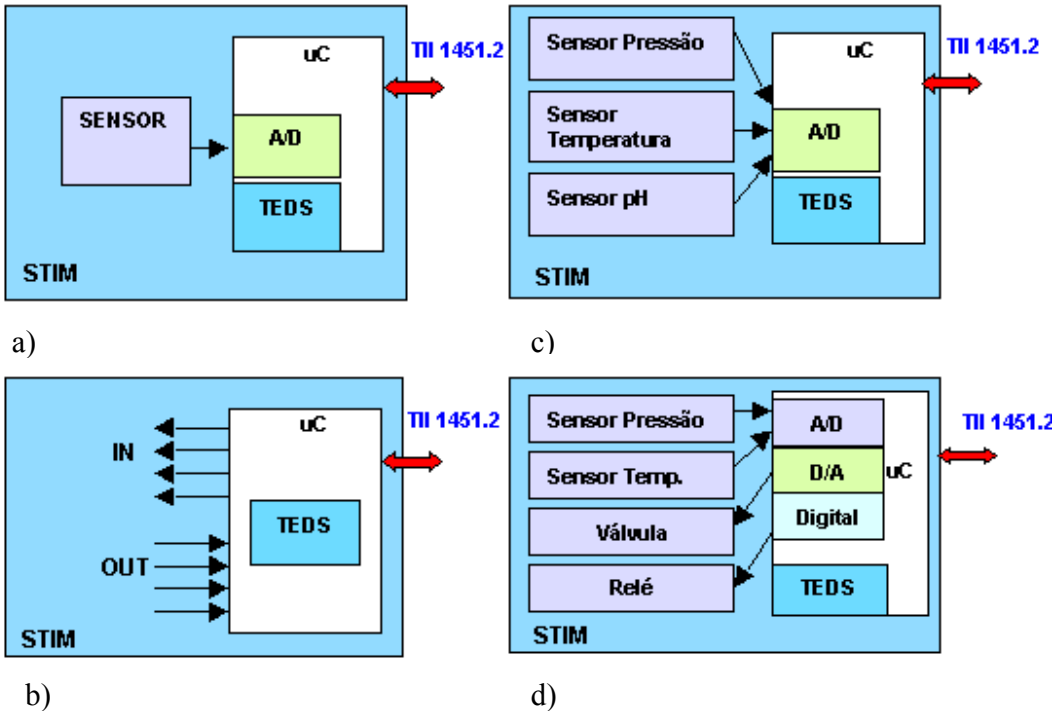


Figura 5.4. Diferentes maneiras de se implementar um STIM: a) STIM dedicado, b) STIM com diversos canais digitais I/O, c) STIM com diversos sensores e d) STIM com diversos sensores e atuadores.

5.4.2 - STIM - *Hardware* Necessário

De acordo com os conceitos estudados, a implementação de um módulo STIM requer, a princípio, um processador e alguns periféricos com suas respectivas interfaces, conforme ilustrado na Figura 5.5.

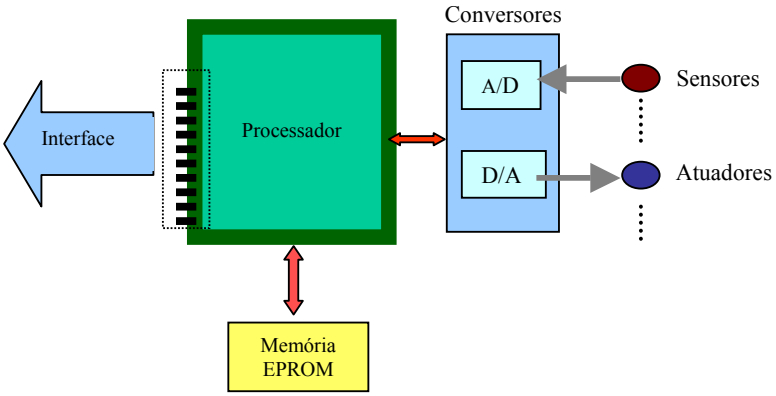


Figura 5.5. *Hardware* necessário à implementação de um STIM.

Os três fatores fundamentais a serem levados em conta na escolha da plataforma de *hardware* para implementar um STIM são os seguintes:

a) Capacidade de suportar a interface digital definida no padrão IEEE 1451.2;

b) Capacidade de suportar a interface com os transdutores;

c) Memória necessária para a implementação das TEDS.

No caso de serem implementados transdutores analógicos, precisa-se de circuitos de condicionamento de sinal e conversores A/D e/ou D/A, para estabelecer a interface com o processador. Para a interface TII com o nó de rede é necessária uma porta de comunicação serial e, finalmente, uma memória não volátil para armazenar as estruturas TEDS. Na Figura 5.6, mostra-se esquematicamente o STIM e seus componentes.

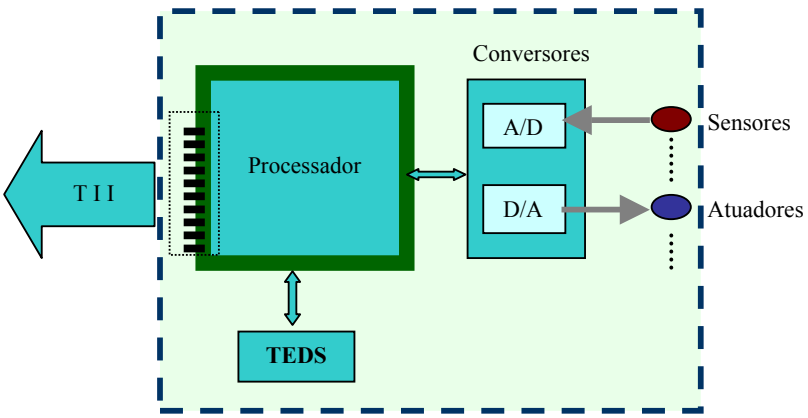


Figura 5.6. Módulo STIM e seus componentes.

Comercialmente existem diversas tecnologias e dispositivos que podem ser utilizados para implementar um STIM como, por exemplo:

a) Sistema de aquisição de dados: Microconversor ADuC812® da *Analog Devices* [24];

b) Microcontroladores da família 6800® da *Motorola* [66];

c) Microcontroladores da família PIC® da *Microchip* [63];

d) Utilização da Tecnologia de Lógica Programável (PLDs).

A alternativa d), utilizando dispositivos PLDs de baixo custo é a escolhida neste trabalho e será abordada no capítulo referente a materiais e métodos.

5.4.3 - Funcionalidade da IEEE 1451.2

A funcionalidade da IEEE 1451.2 refere-se ao conjunto de especificações que dizem respeito ao funcionamento do STIM, independentemente do *hardware* escolhido para implementá-lo. A funcionalidade mencionada abrange funções, endereçamento, disparo, interrupções e transporte de dados. Nesse contexto, chamar-se-á de canal transdutor ao sensor ou atuador e seu canal de comunicação associado diretamente com a lógica IEEE 1451.2, implementada através de um processador.

Um STIM em conformidade com o padrão deverá implementar as seguintes funções:

- Endereçamento;
- Disparo;
- Transporte de dados;
- Controle global;
- Estado global;
- Interrupção e máscaras de interrupção;
- *Hot-swap*;
- Meta-TEDS.

Cada canal transdutor que faz parte do STIM deverá implementar as seguintes funções:

- Canal-TEDS;
- Dados do transdutor;
- Estado;
- Controle.

a) Endereçamento

A função de endereçamento é utilizada em conjunto com a função de transporte de dados via interface TII. A estrutura de endereçamento padronizada faz uso de um conjunto composto por dois *bytes*, chamado de endereço completo. Um endereço completo especifica se o dado deve ser lido ou escrito, com que tipo de função e através de que canal do STIM. A estrutura de um endereço completo divide-se em um *byte* que representa o endereço de função (*byte* mais significativo) e um *byte* que representa o endereço de canal (*byte* menos significativo). A estrutura de um endereço completo é mostrada na Figura 5.7.

Endereço de função *Byte* mais significativo								**Endereço de canal** *Byte* menos significativo							
r/w	Código de função							Número de canal							
msb							lsb	msb							lsb

r =leitura. w= escrita

Figura 5.7. Estrutura de um endereço completo.

A cada canal transdutor componente de um STIM, atribui-se um número entre 1 e 255[16]. Para conhecer a quantidade de canais implementados em um STIM em particular, pode-se acessar o campo "número de canais implementados" pertencente ao bloco de dados Meta-TEDS. O canal zero, denominado de *CHANNEL_ZERO* no padrão, tem um significado especial. Quando uma função é endereçada para o canal zero, refere-se ao STIM como sendo uma coleção de todos os canais. Assim, uma função terá uma interpretação global quando for endereçada para o canal zero e uma interpretação específica para cada canal quando for endereçada para os canais 1 a 255.

A modo de exemplo, o endereço de função 128 utiliza-se para ler dados de transdutores de forma global quando o endereço for aplicado ao canal zero e para ler dados de um transdutor específico quando for aplicado a um canal entre 1 e 255. No apêndice C são apresentados os endereços de função mais utilizados.

[16] O número de canal expressa-se através de um valor de 8 *bits* (256 possibilidades diferentes), sendo que o valor 0 é reservado e os números 1 a 255 caracterizam diferentes canais transdutores.

b) Disparo

O disparo é suportado por uma das linhas da interface TII, especificamente o sinal NTRIG. A função de disparo estabelece o momento em que deve ser enviado um comando desde o NCAP para o STIM (sinal de disparo) através do sinal NTRIG, e o momento em que o STIM deve reconhecer o evento (reconhecimento de disparo) através do sinal NACK, que também constitui uma linha da interface TII. Este tipo de ação fará, por exemplo, com que o um canal transdutor amostre um dado novo. A resposta geral de um STIM a um evento de disparo mostra-se na Figura 5.8.

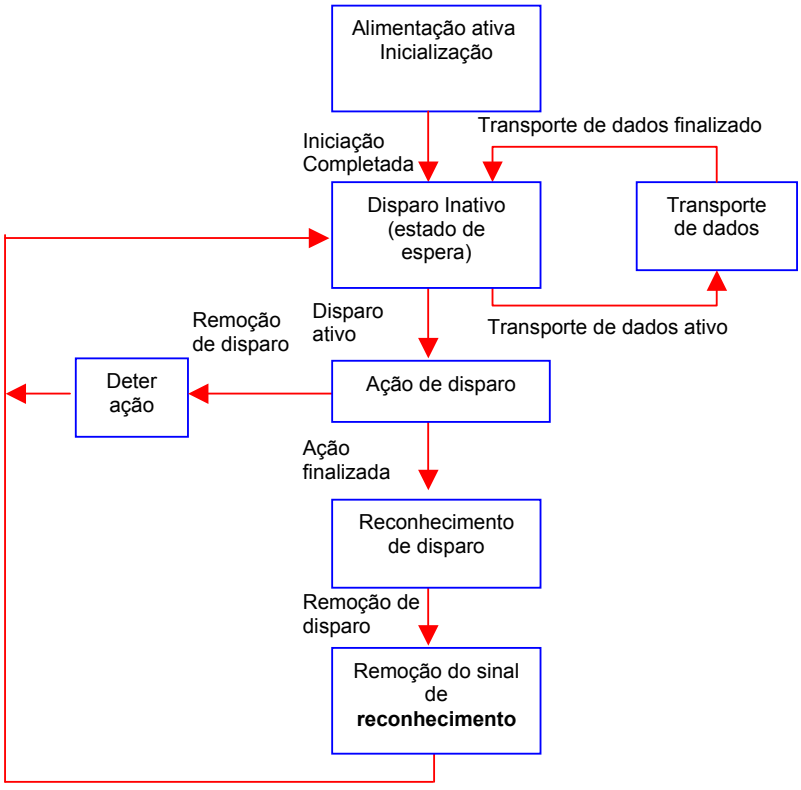


Figura 5.8. Função de disparo.

A primeira etapa do ciclo de disparo é a inicialização do sistema, logo depois, o STIM ficará esperando um evento de disparo. Se o processo de disparo for detido, será abortada toda ação que estiver acontecendo no STIM. Assim sendo, o resultado de um canal transdutor ou o estado de um atuador será indefinido. Se nenhuma ação de detenção acontecer, o STIM reconhecerá o sinal de disparo e ficará esperando até o NCAP mudar o nível lógico do sinal NTRIG. Uma vez feito o

reconhecimento, o STIM removerá esse sinal e o ciclo evoluirá para o estado de espera. O STIM permanecerá nesse estado até completar a transferência de dados (escrita ou leitura de dados de transdutores, estado, controle ou campos de dados TEDS). O disparo permanecerá inativo até o transporte de dados ter finalizado.

c) Transporte de Dados

O transporte de dados se dá através das linhas de sinal da interface padronizada TII. Deste modo é possível identificar quando um sinal de transporte de dados está ativo e, além disso, utilizá-lo para delimitar as estruturas ou quadros de leitura e escrita. O transporte de dados interage com a função de disparo e deve estar inativo antes do sinal de disparo se ativar. Quando a função de transporte de dados passa para o estado ativo, o NCAP envia para o STIM um endereço completo especificando o tipo de operação (escrita ou leitura) a ser desenvolvida, qual a função e qual o canal envolvido no processo. A função de transporte de dados mais comumente usada é a relacionada com os transdutores. Emprega-se para ler dados produzidos por sensores ou escrever dados para serem utilizados por atuadores

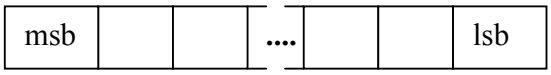
d) Controle

A função de controle permite enviar comandos para o STIM de forma global, ou então, para um canal em particular. Esses comandos irão afetar o estado de funcionamento do STIM. Para escrever um comando de controle, utiliza-se o endereço de função *write channel control command* (escrever comando de controle para um canal), caso for endereçado para um canal entre 1 e 255, ou *write global control command* (escrever comando de controle global), caso for endereçado para o canal zero. Os comandos de controle são compostos por dois *bytes*, e seus valores estão definidos no padrão, para o canal zero e para os canais restantes. Por exemplo, um comando cujo valor é "1" indica *reset* do STIM quando o comando é endereçado para o canal zero e *reset* de canal quando é endereçado para um canal entre 1 e 255. Por seu turno, um comando cujo valor é "0" indica STIM ou canal não operacional.

e) Estado

A função de estado permite ao NCAP, determinar o estado do STIM de forma global ou de um canal em particular. A fim de implementar esta função utilizam-se registradores de estados padrões e auxiliares. Cada *bit* em uma posição específica de um registrador de estado representa a ausência ou presença de uma condição particular. A presença de uma condição é indicada com um *bit* em "1" na posição apropriada. Na Figura 5.9, ilustra-se a idéia de forma esquemática.

A função de estado também é utilizada em conjunto com as máscaras de interrupção e as interrupções para indicar que o STIM está solicitando algum tipo de serviço. Os estados possíveis, tanto padrões quanto auxiliares, são especificados no padrão.



msb: bit mais significativo

lsb: bit menos significativo

Figura 5.9. Registrador de estados.

f) Interrupções

A requisição de interrupção se faz através de uma linha de entrada de uma Unidade Microprocessadora (MPU - *Microprocessor Unit*). A requisição de interrupção é utilizada por dispositivos de entradas e saídas, para interromper a execução do programa em curso e fazer a MPU passar para um programa especial, denominado rotina de serviço de interrupção. Quando a entrada é ativada, a MPU executa esse programa especial, que normalmente envolve atender ao dispositivo que provocou a interrupção. Completada a execução, a MPU retoma o programa em que estava trabalhando ao ser interrompida [67]. Existem situações nas quais responder a uma requisição de interrupção não é conveniente para a MPU. Este é o caso no qual um dispositivo de entradas e saídas interrompe a MPU enquanto ela está executando uma parte de um programa que requer processamento contínuo. Neste caso, a ocorrência de uma interrupção pode resultar indesejável. Por esse motivo, todos os microprocessadores dispõem de algum recurso para inibir a operação de interrupção, sob controle do programa.

A ocorrência da interrupção será ignorada pela MPU quando o sinalizador de inibição de interrupções (*flag*) estiver ativo em uma posição de um registrador de máscaras de interrupção.

Este tipo de interrupção denomina-se mascarável. Outro tipo de interrupção é a não mascarável, neste caso, a MPU sempre dá atenção à requisição independentemente do estado apresentado pelo sinalizador. Em outras palavras, seu efeito não pode ser inibido pelo programador.

O STIM deve conter dois registradores de máscaras de interrupção, um deles padrão e o outro auxiliar. Ambos os registradores devem possuir um tamanho de dois *bytes*. No STIM se realizam operações lógicas do tipo *and*, *bit* a *bit*, entre as posições dos registradores padrão e auxiliar de máscaras de interrupção e os registros padrão e auxiliar de estado. Desta maneira, quando um *bit* estiver em "1" em qualquer uma das posições dos registradores de interrupção e a posição correspondente no registrador de estado também estiver em "1", produz-se uma requisição de interrupção que pode ser global ou para um canal em particular dependendo da posição do *bit* de requisição de serviço, no registrador padrão de estados. Na Figura 5.10, ilustra-se o processo de requisição de interrupção.

A interface digital TII possui uma linha dedicada para possibilitar ao STIM realizar uma requisição de interrupção ao NCAP. Esta linha denomina-se NINT. Cumpre salientar que quem produz a interrupção é o STIM e quem age como MPU é o NCAP. O sinal de interrupção será ativado se a posição correspondente ao *bit* de requisição de serviço (*service request bit*) no registrador padrão de estados do STIM estiver em "1".

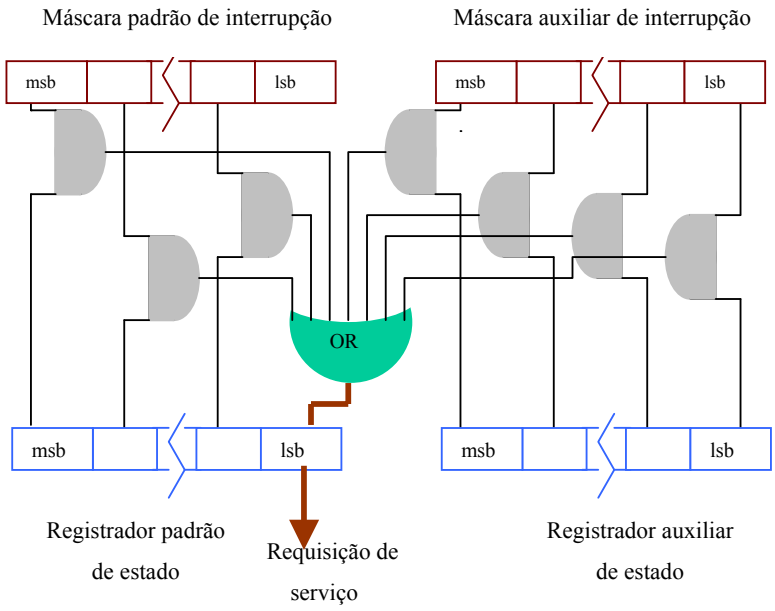


Figura 5.10. Geração de uma requisição de interrupção.

g) Modo "*Hot-Swap*"

O termo *hot-swap* faz referência ao processo de conectar ou desconectar o STIM, sem desligar a alimentação que o NCAP lhe fornece através da interface TII. Isto porque o NCAP é conectado a uma rede e pode não estar habilitado para ser desligado quando se faz necessário desconectar ou conectar o STIM. Deste modo deve ser possível abrir a interface TII, enquanto o NCAP continua a ser alimentado. A interface fornece um sinal para detectar eventos de inserção (quando um STIM é conectado a um NCAP com alimentação ativa) ou remoção (quando um STIM é retirado de um NCAP com alimentação ativa). Este sinal denomina-se NSDET.

h) Meta-TEDS

Este assunto será tratado na seção 5.4.7.

5.4.4 - Interface Independente de Transdutores (TII)

A interface TII é o vínculo físico entre o STIM e o NCAP. A TII suporta dez linhas de sinal, através das quais estabelece-se a comunicação entre os módulos e a transferência de dados sob comunicação série do tipo *half duplex*.

a) Aspectos Físicos e Elétricos da TII

A TII é composta por dez pinos sendo que oito deles suportam a funcionalidade do padrão e os dois restantes fornecem a alimentação. A TII e seus sinais são mostrados na Figura 5.11.

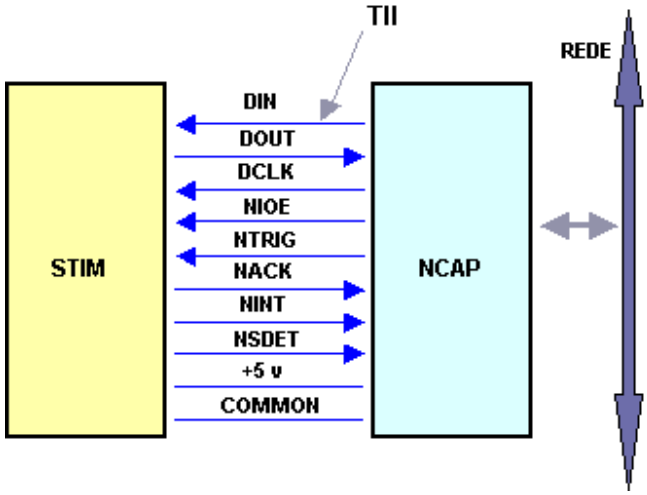


Figura 5.11. Interface TII entre STIM e NCAP.

A comunicação através da TII é do tipo série síncrona e concebida com base no protocolo *Serial Peripheral Interface* (SPI™)[17]. A seguir é apresentada a Tabela 5.1, onde estão definidas as linhas da interface, indicando-se o tipo de lógica utilizada e qual o módulo que opera cada sinal.

Na Tabela 5.2 é exibida a atribuição de pinos e cores, recomendada pelas especificações do padrão.

Tabela 5.1. Definição dos sinais da Interface Independente de Transdutores (TII).

Grupo	Sinal	Abreviação	Lógica	Operada pelo:	Descrição
Dados	DATA_OUT	DOUT	Positiva	STIM	Transporte de dados desde o STIM para o NCAP
	DATA_IN	DIN	Positiva	NCAP	Endereçamento e transporte de dados desde o NCAP para o STIM
	DATA_CLOCK	DCLK	Borda de subida	NCAP	A cada evento de relógio são guardados os dados associados a DIN ou DOUT
	N_IO_ENABLE	NIOE	Ativo baixo	NCAP	Sinaliza se o transporte de dados está ativo e delimita a estrutura de dados transportados
Disparo	N_TRIGGER	NTRIG	Borda de descida	NCAP	Executa funções de disparo
Suporte	POWER	POWER	-	NCAP	Alimentação + 5 V
	COMMON	COMMON	-	NCAP	Referência
	N_ACKNOWLEDGE	NACK	Borda de descida	STIM	Serve para duas funções: 1.Reconhecimento de disparo 2.Reconhecimento de transporte de dados
	N_STIM_DETECT	NSDET	Ativo baixo	STIM	Usada pelo NCAP para detectar a presença do STIM
Interrupção	N_IO_INTERRUPT	NINT	Borda de descida	STIM	Usada pelo STIM para solicitar serviço ao NCAP

[17] *Serial Peripheral Interface* (SPI) é uma marca registrada pela Motorola. O SPI é um barramento utilizado na comunicação série síncrona bidirecional entre microcontroladores e seus periféricos.

Tabela 5.2 TII; Atribuição de pinos e cores.

Pino	Sinal	Cor	Direção para o NCAP	Direção para o STIM
1	DCLK	Marrom	Saída	Entrada
2	DIN	Vermelho	Saída	Entrada
3	DOUT	Laranja	Entrada	Saída
4	NACK	Amarelo	Entrada	Saída
5	COMMON	Verde	Alimentação	Alimentação
6	NIOE	Azul	Saída	Entrada
7	NINT	Roxo	Entrada	Saída
8	NTRIG	Cinza	Saída	Entrada
9	POWER	Branco	Alimentação	Alimentação
10	NSDET	Preto	Entrada	Saída

Fonte: Std. IEEE 1451.2. Adaptado.

Com a finalidade de definir as entradas e saídas, utiliza-se o STIM como referência. O sinal DIN é a entrada de dados no STIM e DOUT é a saída de dados do mesmo módulo. A mesma convenção utiliza-se para os outros sinais. O sinal DCLK é o relógio operado pelo NCAP. O sinal NIOE faz referência à habilitação de entrada / saída e também é controlado pelo NCAP. Através do sinal NTRIG, o NCAP efetua um evento de disparo, e por meio do NACK, o STIM envia um sinal de reconhecimento. A linha NINT é utilizada pelo STIM para efetuar uma requisição de serviço e, o NSDET é utilizado para detectar a presença do STIM. Finalmente são fornecidos, a alimentação necessária de 5 V e o pino de referência.

O NCAP é quem controla todas as comunicações. Na verdade, a única linha que pode ser operada diretamente pelo STIM é a linha NINT. Embora apresentadas no padrão como operadas pelo STIM, as linhas DOUT, NSDET e NACK, não são totalmente controladas pelo STIM. A transferência de dados através de DIN se dá apenas quando o NCAP o requisitar, NSDET é um sinal passivo que se conecta à terra e a linha NACK só é ativada em resposta a uma ação do NCAP [64].

Quanto ao conector físico para implementar a interface, não se faz referência no padrão. O grupo de trabalho do IEEE 1451.2 tentou padronizar um tipo de conector, porém foi descoberto que os conectores são muito dependentes da aplicação para poder estabelecer um consenso geral.

Alguns tipos de conectores já têm sido utilizados com sucesso. Conectores tipo *latch* com 10 terminais (5x2) para cabo plano (*flat*) têm sido utilizados para demonstrações em ambientes pouco agressivos. Outros conectores como o *D shell*[18], de 15 pinos, usado normalmente em sistemas informatizados, provavelmente sejam mais apropriados para aplicações industriais.

Outro aspecto importante é a forma em que é alimentado o módulo STIM. O NCAP deve fornecer + 5 V +/- 0,20 V de corrente contínua, via linha POWER da interface TII. O consumo do STIM não deve superar os 75 mA para o valor de tensão especificado.

5.4.5 - Protocolos de Comunicação

Os protocolos especificados no padrão IEEE 1451.2 descrevem a implementação da função de disparo e a forma em que se realiza a transferência de dados utilizando a interface TII. Fala-se em gravação ou operação de escrita quando o NCAP envia informações em formato de quadro através da linha DIN. Essas informações são caracterizadas por um *byte* indicando um endereço de função, um *byte* indicando o endereço de canal correspondente e um ou mais *bytes* de dados a serem escritos. Fala-se em leitura quando o STIM envia informações em formato de quadro através da linha DOUT. Essas informações são dados associados a um sensor, enviadas para o NCAP e lidas através da rede.

A seguir são apresentados os protocolos de comunicação especificados no padrão IEEE 1451.2 e que foram implementados no presente trabalho.

a) Protocolo de Disparo

A função de disparo é utilizada, por exemplo, quando se deseja ler os dados de um canal transdutor do STIM. A seqüência apresentada a seguir ilustra-se na Figura 5.12.
a) NCAP espera durante um intervalo de tempo igual a *Channel Write Setup Time* (tws), gravado no bloco de dados Canal-TEDS.
b) NCAP ativa a linha NTRIG.
c) STIM reconhece o disparo ativando o sinal NACK (ativo baixo).
d) NCAP muda o estado lógico do sinal NTRIG.

[18] *D-Shell* é um tipo de conector padronizado, conhecido também como conector tipo DB. A pinagem do conector fica dentro de uma estrutura em forma de D, daí o nome *D Shell*.

e) STIM muda o estado lógico do sinal NACK.

f) NCAP espera durante um intervalo de tempo igual a *Channel Read Setup Time* (trs)[19], gravado no bloco de dados Canal-TEDS.

Por simplicidade, as transições no diagrama de tempos têm sido consideradas verticais, embora, na realidade, correspondam a uma rampa ascendente ou descendente devido ao retardo próprio dos sistemas digitais.

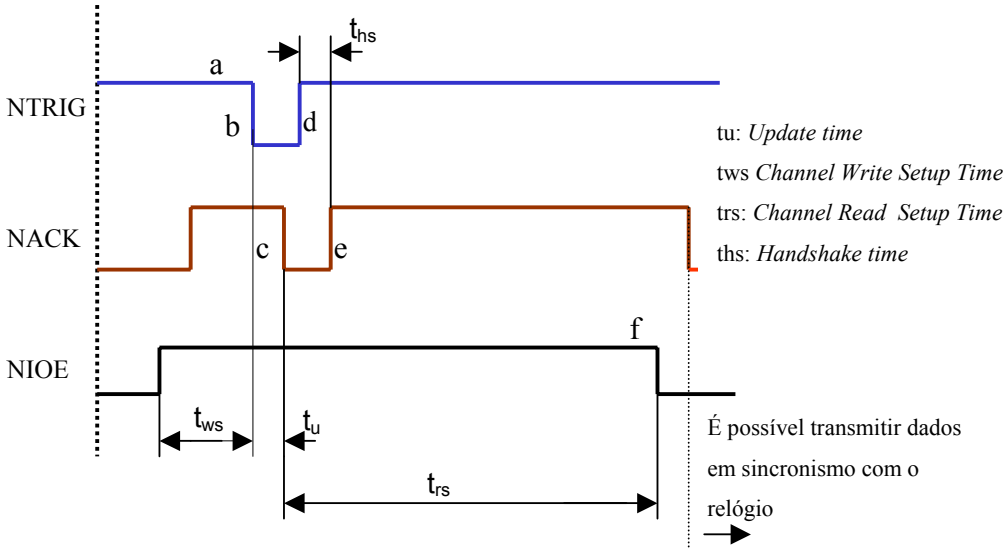


Figura 5.12. Protocolo que implementa a função de disparo.

Após um evento de disparo e, estando o sinal NIOE ativo, assim que ocorre uma transição na linha NACK é possível a transferência de *bits*. A partir desse momento, e em sincronismo com o sinal de relógio (DCLK) é possível realizar operações de escrita e de leitura utilizando-se as linhas DIN e DOUT da interface (protocolos de transferência de quadro de leitura e escrita).

b) Transferência de *Bits*

A transferência em série de *bits* entre o NCAP e o STIM realiza-se via linha DIN, e via DOUT, desde o STIM para o NCAP. A transferência de dados é controlada pelo sinal de relógio (DCLK) seguindo a seguinte seqüência:

a) Evento na linha DCLK (borda de subida).

[19] Os tempos que aparecem no diagrama da Figura 5.12 são especificados nos formatos TEDS (ver 5.4.6).

b) Na primeira borda de descida do sinal DCLK, o primeiro *bit* é enviado pelo emissor (NCAP via DIN, ou STIM via DOUT).
c) Na seguinte borda ascendente, o *bit* é armazenado pelo receptor (NCAP em DIN ou STIM em DOUT).
d) Os *bits* seguintes são transmitidos repetindo-se os passos b) e c).
A comunicação é do tipo *half duplex,* portanto, quando for estabelecida a comunicação entre o NCAP e o STIM, a linha DOUT é ignorada pelo NCAP, e vice-versa.

c) Protocolo de Transferência de Quadro de Leitura

A transferência entre os módulos deve ser feita em pacotes de 8 *bits*, utilizando-se o protocolo de transferência de *bits*. O NCAP começará a transferir um *byte* de escrita ou leitura só depois de observar uma transição na linha NACK. O módulo STIM estabelecerá uma transição na linha NACK quando for apropriadamente entregue o *byte* prévio e o NCAP estiver pronto para proceder. Este fato é sinalizado com asteriscos na Figura 5.13. Tanto o protocolo de transferência de quadro de leitura como o de escrita são apresentados na Figura 5.13. A seqüência para transferir um quadro de leitura é a seguinte:
a) NCAP ativa o sinal NIOE (A).
b) NCAP espera até o STIM ativar o sinal NACK (B).
c) NCAP escreve o endereço de função (1 *byte*) (C).
d) NCAP escreve o endereço do canal correspondente (1 *byte*) (D).
e) NCAP lê os dados desde o *byte* mais significativo até o *byte* menos significativo (E).
f) NCAP muda o estado lógico do sinal NIOE (F).
g) STIM muda o estado lógico do sinal NACK (G).

d) Protocolo de Transferência de Quadro de Escrita

A seqüência para transferir um quadro de escrita é a seguinte:
a) NCAP ativa o sinal NIOE.
b) NCAP espera até o STIM ativar o sinal NACK.
c) NCAP escreve o endereço de função (1 *byte*).
d) NCAP escreve o endereço do canal correspondente (1 *byte*).
e) NCAP escreve os dados desde o *byte* mais significativo até o *byte* menos significativo.

f) NCAP muda o estado lógico do sinal NIOE.

g) STIM muda o estado lógico do sinal NACK.

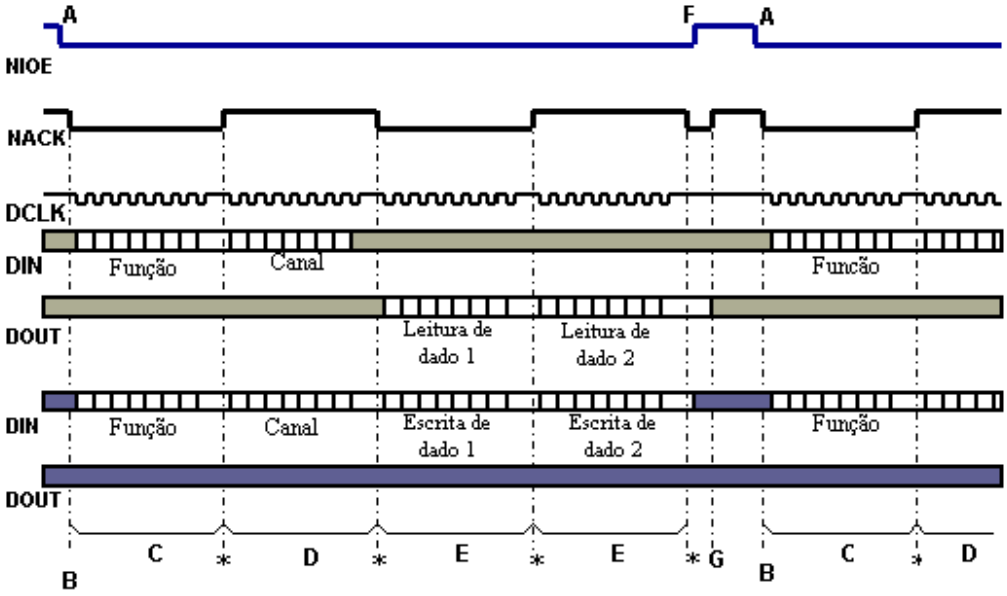


Figura 5.13. Protocolos de transferência de quadro de leitura e escrita.

Fonte: Std. IEEE 1451.2. Adaptado.

5.4.6 - Formatos TEDS

Provavelmente o aspecto mais interessante do padrão IEEE 1451.2, diz respeito à utilização dos formatos TEDS. As TEDS são as especificações de transdutores em formato eletrônico, analisadas a seguir com mais detalhes. Quando se faz necessário implementar algum tipo de transdutor para uma determinada aplicação é desejável e necessário conhecer as especificações do dispositivo. É assim, que normalmente recorre-se à folha de dados do dispositivo (*data sheet*). Na folha de dados encontram-se diversas informações que caracterizam o transdutor quanto a seu desempenho e sua identificação. Deste modo, disponibiliza-se informações relevantes como características de entrada/saída, valores máximos e mínimos, resolução, especificações de erro e outras informações tais como, fornecedor, número de série e versão. Se essas informações pudessem estar disponíveis em formato digital, obedecendo a uma forma padronizada e ficassem

disponíveis para serem lidas através da rede, seria resolvido um conjunto de problemas associados ao interfaceamento de transdutores com redes de controle e barramentos de campo. Seriam facilitadas as tarefas de manutenção bem como o trabalho com sistemas que utilizam diversas entradas/saídas. Nesses aspectos baseia-se a idéia de "dados em formato eletrônico".

Assim, tentando implementar as idéias citadas acima e visando resolver os problemas de interfaceamento entre transdutores e rede, foi criado o conceito TEDS do padrão IEEE 1451.2.

As TEDS são as especificações do transdutor, equivalente àquelas que se encontrariam no manual do fabricante, escritas em formato eletrônico e armazenadas em um elemento de memória não volátil.

Quando é feita uma solicitação por parte do NCAP, o processador do STIM acessa essas informações e as disponibiliza para a rede, via interface padronizada TII.

Nas décadas passadas, a integração de transdutores e microcontroladores fez com que o conceito de transdutor inteligente começasse a ser realidade. Na verdade, as vantagens apresentadas por essa configuração não foram totalmente aproveitadas. Hoje, a inclusão dos formatos TEDS nessa integração adiciona o conceito de auto-identificação, uma ferramenta de importância fundamental para o modo *plug and play* ser implementado. Esta, junto à utilização de uma interface padronizada para estabelecer a comunicação com o nó de rede baseado em microprocessador, é a grande contribuição do padrão IEEE 1451.2 na área de sistemas de instrumentação eletrônica.

5.4.7 - Especificação dos Formatos TEDS

O padrão possui uma cláusula específica que define os formatos TEDS. As especificações estabelecem diretrizes quanto aos formatos lógicos e ao conteúdo das TEDS, mas não se faz restrição nenhuma quanto à forma física das TEDS. Porém, é necessário que se mantenham fisicamente associadas ao transdutor que descreve, durante o funcionamento normal do STIM conectado ao NCAP e, também, quando o STIM é retirado do NCAP para propósitos tais como transporte ou calibração.

A definição das estruturas TEDS demandou a maior parte do esforço do grupo IEEE 1451.2, colocando-se especial ênfase em tratar de identificar todos os usos razoáveis dos campos de dados incluídos nos formatos e, além disso, fazendo certas previsões, levando em conta as possíveis expansões futuras dos formatos de acordo com as necessidades da indústria. Cada

estrutura TEDS inclui a informação de seu próprio tamanho em *bytes* e uma soma de verificação (*checksum*) a fim de detectar erros de comunicação.

O padrão define oito formatos TEDS diferentes, sendo que dois desses formatos, Canal-TEDS e Meta-TEDS são obrigatórios e os seis restantes são opcionais.

Na tabela 5.3 é apresentada a estrutura de dados Canal-TEDS, tal como pode ser encontrada no padrão. Cumpre salientar que foi respeitada a terminologia em inglês para ficar em conformidade com o padrão. Através do bloco Canal-TEDS é caracterizado um canal em particular entre 1 e 255, definindo: o tipo de transdutor, os limites máximos e mínimos, as unidades físicas SI, as restrições de tempo e outras informações necessárias para descrever cada canal transdutor implementado no STIM. No caso do STIM conter mais do que um canal transdutor, a estrutura Canal- TEDS deve ser repetida, uma para cada canal.

O bloco de dados Canal-TEDS é acessado pelo NCAP, através do endereço de função 160, aplicado aos canais 1 a 255.

Na Tabela 5.4 apresenta-se a estrutura de dados Meta-TEDS que contém dados que descrevem o módulo STIM de forma globa1, como sendo um conjunto de transdutores, daí o nome Meta. Meta é um prefixo de origem grega que significa “aquele que pertence a uma entidade global ou que possui alguma coisa em comum ou em combinação com todos os membros que formam o conjunto”.

O bloco de dados Meta-TEDS é acessado pelo NCAP, através do endereço de função 160, aplicado ao canal 0.

b) Tipos de Dados

Os tipos de dados têm uma cobertura extensa no padrão. Por exemplo, F32L indica número de ponto flutuante (*float*) precisão simples, de acordo com o padrão IEEE Std 754-1985. O valor correspondente é representado por meio de 32 *bits*. A letra L associada é utilizada para caracterizar tamanho (*length*). Para maiores detalhes sobre os tipos de dados pode-se recorrer às referências [2] e [3].

Tabela 5.3. Bloco de Dados Canal-TEDS.

Field nº	Description	Type	Nº of bytes
Data structure related data sub-block			
1	Channel TEDS Length	U32L	4
2	Calibration Key	U8E	1
3	Channel Industry Calibration TEDS Extension Key	U8E	1
4	Channel Industry Nonvolatile Data Fields Extension Key	U8E	1
5	Channel Industry TEDS Extension Key	U8E	1
6	Channel End-User's Application-Specific TEDS Key	U8E	1
7	Channel Writable TEDS Length	U32C	4
Transducer related data sub-block			
8	Channel Type Key	U8E	1
9	Physical Units	UNITS	10
10	Lower Range limit	F32	4
11	Upper Range Limit	F32	4
12	Worst-Case Uncertainty	F32	4
13	Self-Test Key	U8E	1
Data converter related data sub-block			
14	Channel Data Model	U8E	1
15	Channel Data Model Length	U8C	1
16	Channel Model Significant Bits	U16C	2
17	Channel Data Repetitions	U16C	2
18	Series Origin	F32	4
19	Series Increment	F32	4
20	Series Unit	UNITS	10
Timing related data sub-block			
21	Channel Update Time (t_u)	F32	4
22	Channel Write Setup Time (t_{ws})	F32	4
23	Channel Read Setup Time (t_{rs})	F32	4
24	Channel Sampling Period (t_{sp})	F32	4
25	Channel Warm-Up Time	F32	4
26	Channel Aggregated Hold-Off-Time (t_{ch})	F32	4
27	Timing Correction	F32	4
28	Trigger Accuracy	F32	4
Event sequence options field			
29	Event Sequence Options	U8E	1
Data Integrity data sub-block			
30	Checksum for Channel TEDS	U16C	2

Fonte: Std. IEEE 1451.2.

Tabela 5.4. Bloco de Dados Meta-TEDS.

Field nº	Description	Type	Nº of bytes
TEDS version constant related data sub-block			
1	Meta-TEDS Length	U32L	4
2	IEEE 1451 Standards Family Working Group Number	U8E	1
3	TEDS Version Number	U8E	1
Identification related data sub-block			
4	Globally Unique Identifier	UUID	10
Data structure related data sub-block			
5	CHANNEL_ZERO Industry Calibration TEDS Extension Key	U8E	1
6	CHANNEL_ZERO Industry Nonvolatile Data Field Extension Key	U8E	1
7	CHANNEL_ZERO Industry TEDS Extension Key	U8E	1
8	CHANNEL_ZERO End-User's Application-Specific TEDS Key	U8E	1
9	Number of implemented channels	U8C	1
10	Worst-Case Channel Data Model Length	U8C	1
11	Worst-Case Channel Data Repetitions	U16C	2
12	CHANNEL_ZERO writable TEDS Length	U32C	4
Timing related data sub-block			
13	Worst-Case Channel Update Time (t_{wu})	F32	4
14	Global Write Setup Time (t_{gws})	F32	4
15	Global Read Setup Time (t_{grs})	F32	4
16	Worst-case Channel Sampling Period (t_{wsp})	F32	4
17	Worst-case Channel Warm-Up Time	F32	4
18	Command Response Time	F32	4
19	STIM Handshake Timing (t_{hs})	F32	4
20	End-of-Frame Detection Latency (t_{lat})	F32	4
21	TEDS Hold-Off Time (t_{th})	F32	4
22	Operational Hold-Off Time (t_{oh})	F32	4
23	Maximum Data Rate	U32C	4
Channel grouping related data sub-block			
24	Channel Groupings Data Sub-Block Length	U16L	2
25	Number of Channel Groupings = G	U8C	1
Fields 26-28 are repeated G times, once for each group			
26	Group Type	U8E	1
27	Number of Group Members = N	U8C	1
28	Member Channel Numbers List = M(N)	Array of U8E	N
Data integrity data sub-block			
29	Checksum for Meta-TEDS	U16C	2

Fonte: Std. IEEE 1451.2.

5.5 - O Padrão IEEE 1451.1

O padrão IEEE 1451.1 foi criado com o objetivo de padronizar o *software* da interface entre NCAPs e redes de controle. A padronização é atingida através do desenvolvimento de um conjunto comum de informações, conhecido como modelo orientado a objeto, que inclui blocos de transdutores, blocos de função e blocos de rede [2].

Neste sentido, o principal objetivo do NIST/IEEE é o desenvolvimento de métodos de conexão padronizados entre transdutores inteligentes e redes de controle, utilizando as tecnologias já existentes. A idéia é que ao utilizar diferentes métodos de conversão A/D, diferentes microprocessadores, ou diferentes protocolos e *drivers* de rede, seja feita a menor quantidade possível de mudanças no sistema [5]. Dito em outras palavras, o padrão IEEE 1451.1 tenta especificar a interface com a rede de uma maneira tal que a aplicação independa do protocolo utilizado pela rede ou barramento de campo.

Para atingir esses objetivos faz-se necessário o emprego dos recursos da programação orientada a objetos. Assim é possível definir classes e objetos para o transdutor e para o transdutor conectado a uma rede, junto à especificação da interface para esses modelos.

Um NCAP em conformidade com o padrão IEEE 1451.1 pode funcionar com o módulo STIM conectado através da TII, como definido no padrão IEEE 1451.2, e pode funcionar de acordo com as especificações das diretrizes IEEE 1451.3 e IEEE 1451.4.

No entanto, cumpre salientar que o interesse fundamental do presente trabalho centra-se no NCAP funcionando com o módulo STIM, portanto, entende-se que toda referência feita sobre transdutores, refere-se ao módulo STIM conectado através da interface TII.

5.5.1 - Processador de Aplicação com Capacidade de Operar em Rede (NCAP)

O conceito de NCAP tal como apresentado no padrão é muito abrangente o que acarreta certa dificuldade de entendimento. Em termos simples, o NCAP é um nó com capacidade de processamento local, que é "visto" através da rede como sendo um ponto de inteligência distribuída. Porém, distingue-se de um sistema convencional processador- transmissor por dois motivos fundamentais que constituem a base do padrão:

a) O NCAP consegue se abstrair do tipo de rede ao qual é conectado, introduzindo assim o conceito de interoperabilidade;

b) O NCAP consegue se abstrair do tipo de transdutor conectado (aplicação), facilitando a introdução do modo de operação *plug and play*.

O *software* do NCAP é baseado em um modelo de informação que pode ser descrito através dos recursos da programação orientada a objeto.

Na Figura 5.14, ilustra-se o conceito de NCAP, apresentando os componentes principais como definidos no padrão. O modelo do NCAP é dividido em dois pontos de vista diferentes, o ponto de vista físico, esquematizado com linhas cheias na Figura 5.14, e o ponto de vista lógico, esquematizado com linhas tracejadas e pontilhadas.

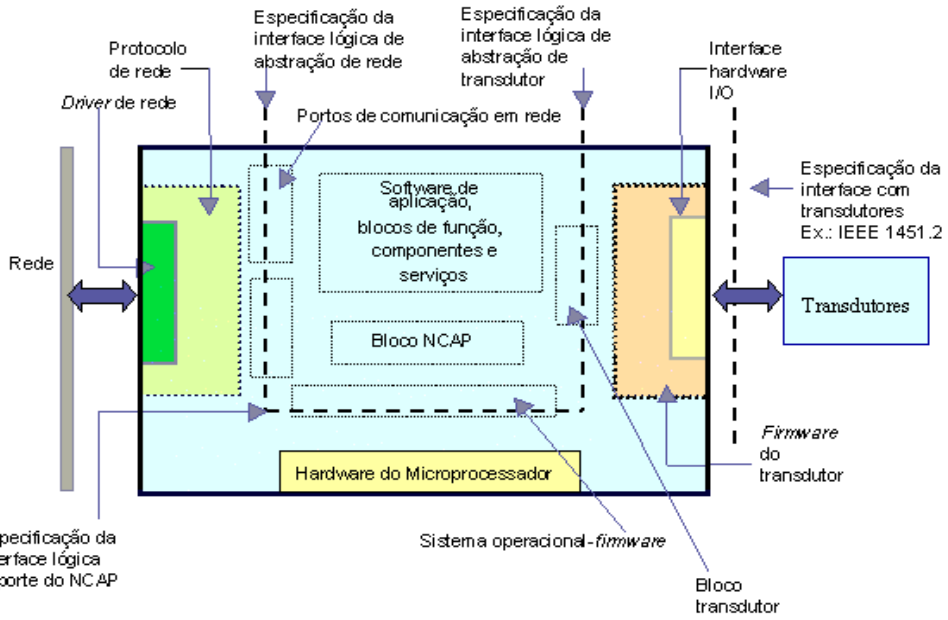


Figura 5.14. Diagrama do NCAP.

a) Ponto de Vista Físico

Esta visão do modelo abrange os componentes físicos do sistema, ou seja, o *hardware* do NCAP, que é composto de um microprocessador e seus circuitos associados, o *driver* de rede e a interface de *hardware* de entradas/saídas para transdutores.

b) Ponto de Vista Lógico

Esta visão do modelo, que constitui o "coração" do conceito NCAP, abrange os componentes lógicos que podem ser agrupados em componentes de aplicação e de suporte.

Os componentes de suporte são o sistema operacional, o protocolo de rede é o *firmware*[20] dos transdutores. O sistema operacional fornece uma interface para as aplicações que se denomina "interface lógica suporte do NCAP".

Uma outra interface lógica, chamada de "especificação da interface lógica de abstração de rede" é composta por objetos de *software* chamados de portos, fornecendo o nível de abstração necessário a fim de ocultar detalhes da comunicação com a rede, através de um pequeno conjunto de métodos de comunicação.

A terceira interface lógica, chamada de "especificação da interface lógica de abstração de transdutor" é composta por objetos *software* agrupados no bloco transdutor, fornecendo o nível de abstração necessário, visando ocultar detalhes da comunicação com a aplicação que contém o transdutor.

Finalmente, os componentes de aplicação são modelados como sendo blocos de função em combinação com componentes e serviços. Deste conjunto também faz parte o bloco NCAP.

c) NCAP Visão Conceitual

O NCAP pode ser analisado como sendo um bloco central (bloco NCAP) onde são associados diferentes objetos de *software*. As comunicações de rede são "vistas" como portos pelo NCAP. O transdutor é mapeado no NCAP utilizando o objeto bloco transdutor, via interface IEEE 1451.2. Incluem-se também, blocos de função e objetos como parâmetros e ações [68]. Na Figura 5.15 apresenta-se a visão conceitual de um NCAP.

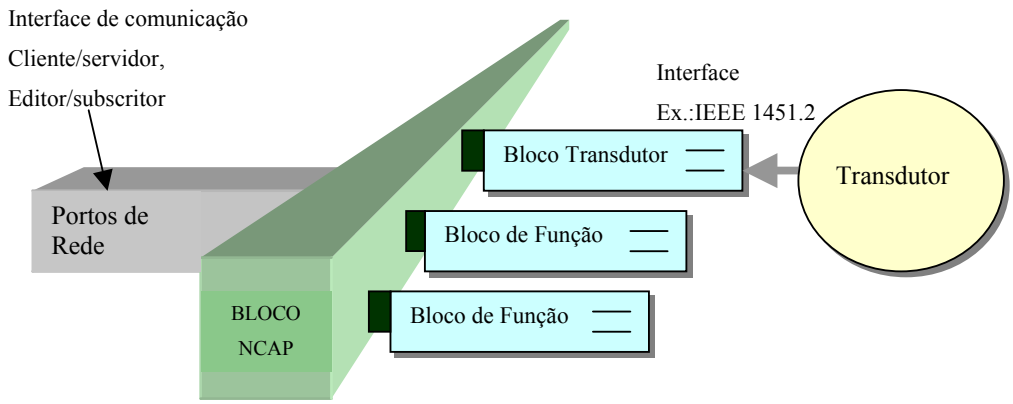


Figura 5.15. Visão conceitual de um NCAP.

[20]*Firmware:* Programa que pode ser armazenado de modo permanente ou semipermanente. Por exemplo, o *software* elementar responsável pela inicialização de um processador e seus circuitos de interface.

Em uma aplicação mínima IEEE 1451.1 devem ser considerados os seguintes elementos:
a) O Bloco NCAP, responsável pelo gerenciamento do sistema;
b) O Bloco Transdutor, que fornece a conexão *software* com o dispositivo transdutor;
c) O Bloco de função, que fornece o algoritmo de aplicação para o transdutor;
d) Parâmetros, que contêm as variáveis acessíveis através da rede;
e) Portos, que são os objetos de *software* necessários para que possa existir uma interação entre NCAPs, utilizando-se o modelo de comunicação cliente/servidor ou o modelo editor/subscritor.

5.5.2 - O Modelo de Informação IEEE 1451.1

O modelo de informação constitui a base da especificação IEEE 1451.1. Através da modelagem de um dispositivo transdutor em termos da programação orientada a objetos, introduz-se uma visão abstrata das características do dispositivo. O modelo é geral o suficiente para abranger uma ampla variedade de serviços relacionados com a conexão de dispositivos transdutores em ambientes de rede [20].

No padrão IEEE 1451.1 especifica-se uma arquitetura de *software* aplicável em sistemas distribuídos, constituídos por um ou mais NCAPs se comunicando através de uma rede.

A arquitetura de *software* é definida a partir de três tipos de modelos [2]:
a) Um modelo orientado a objeto, para os componentes de *software* dos sistemas IEEE 1451.1;
b) Um modelo de dados, para a informação que é comunicada através da rede;
c) Dois modelos de comunicação de rede, o cliente/servidor e o editor/subscritor.

a) Modelo Objeto

Através deste modelo são especificados os tipos de componentes de *software* utilizados para projetar e implementar aplicações em conformidade com o padrão IEEE 1451.1. A hierarquia de classes[21] IEEE 1451.1 é apresentada na Figura 5.16.

[21] A fim de interpretar o conceito de classe é necessário o conhecimento das bases da programação orientada a objetos. Alguns conceitos elementares encontram-se no Capítulo 6.

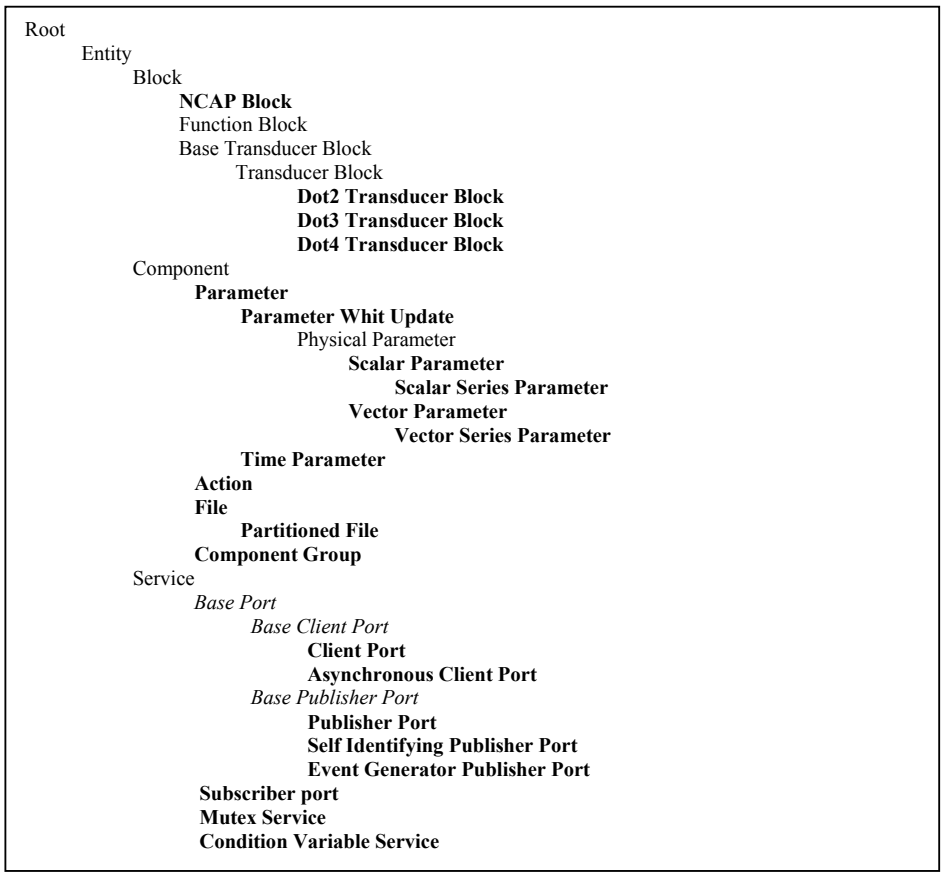


Figura 5.16. Hierarquia de Classes IEEE.

Fonte: Std. IEEE 1451.1.

Tome-se como exemplo o conjunto de classes bloco (*Block Classes*). Nesse caso têm-se:

a1) Classe `IEEE1451_NCAPBlock`, que fornece uma interface *software* padrão para suportar as comunicações de rede e a configuração do sistema.

a2) Classe `IEEE1451_BaseTransducerBlock`, que fornece uma interface *software* padrão entre transdutores e funções de aplicação no NCAP.

a3) Classe `IEEE1451_FunctionBlock`, que realiza o encapsulamento de uma aplicação específica.

b) Modelo de Dados

Através do modelo de dados especifica-se o tipo e a forma da informação comunicada através de uma interface objeto, de forma remota ou local. Por exemplo, dados do tipo primitivo como

boolean, *octet*, *integer*, *float* e *string*. O padrão faz uma ampla cobertura dos tipos de dados primitivos e como eles podem ser utilizados.

c) Modelos de Comunicação em Rede

No padrão IEEE 1451.1 são fornecidos dois modelos de comunicação entre objetos em um sistema distribuído. O modelo cliente/servidor e o modelo editor/subscritor. O primeiro deles é de interesse no presente trabalho.

No modelo cliente-servidor, os usuários são chamados de clientes e a comunicação se dá geralmente através de uma mensagem de solicitação do cliente enviada para o servidor, pedindo para que alguma tarefa seja executada. Após isto, o servidor executa a tarefa e envia a resposta. Essas ações são chamadas de serviços. Geralmente, há muitos clientes usando um pequeno número de servidores [33]. O modelo é ilustrado na Figura 5.17.

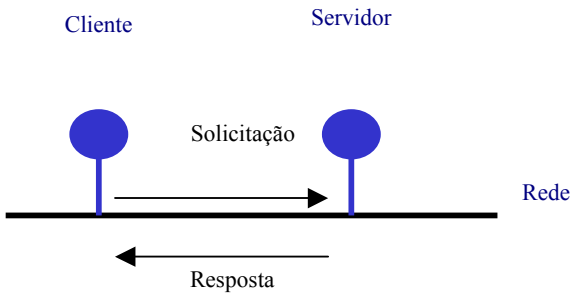


Figura 5.17. Modelo cliente-servidor.

5.6 – Comentários Finais sobre o Capítulo 5

Abordados os conceitos fundamentais relacionados com as diretrizes IEEE 1451.1 e IEEE 1451.2, conclui-se que um STIM pode ser implementado através de um microcontrolador, de soluções comerciais e de dispositivos lógicos programáveis, ao passo que o NCAP deve ser suportado por um processador e um *software* desenvolvido em uma linguagem orientada a objeto.

No capítulo seguinte serão tratadas as tecnologias empregadas no trabalho para implementar um nó 1451 para aplicações em ambientes de instrumentação distribuída sob o modelo de comunicação cliente-servidor.

Capítulo 6

Tecnologias Utilizadas no Projeto

Neste capítulo serão abordadas as tecnologias empregadas na implementação do nó IEEE 1451. Na seção 6.1 trata-se da tecnologia *Ethernet,* abordando-se conceitos elementares, a camada de enlace de dados, a camada física e uma discussão sobre a utilização da *Ethernet* na automação industrial. A *Ethernet* foi empregada neste projeto, em nível de rede de controle, interligando um NCAP servidor com um NCAP cliente. Na seqüência serão analisados conceitos específicos relacionados com a tecnologia de lógica programável, abrangendo-se dispositivos lógicos programáveis, linguagens de descrição de *hardware* e ambientes de síntese. Os dispositivos mencionados e a linguagem VHDL foram empregados no desenvolvimento do STIM e parte do NCAP, implementados neste trabalho.

Finalmente serão abordados conceitos básicos da tecnologia Java, a qual foi empregada no desenvolvimento do *software* que compõe a parte lógica do NCAP implementado. Em particular, serão abordadas as APIs RMI, COMM, e NET.

6.1 – Tecnologia *Ethernet*

Atualmente, *Ethernet* é uma tecnologia de LAN com capacidade de transmitir informação entre computadores a velocidades de 10 a 1000 Mbps. A versão mais comumente utilizada é a de 100 Mbps, empregando cabo de par trançado como meio físico de transmissão.

Numa rede com tecnologia *Ethernet*, todas as estações são conectadas ao mesmo meio físico (barramento). As estações podem estabelecer uma transmissão no momento em que quiserem, entretanto, se houver uma colisão de dois ou mais pacotes de informação, cada computador aguardará um tempo aleatório e fará uma nova tentativa. Este mecanismo de acesso ao meio é conhecido como Acesso Múltiplo com Detecção de Portadora, com Detecção de Colisão (CSMA/CD – *Carrier Sense Multiple Access with Collision Detection*). Para enviar dados, uma estação primeiro “escuta o meio”; se o canal de comunicação estiver livre, então, inicia a

transmissão; se "sentir" colisão, espera um tempo aleatório crescente e reinicia o algoritmo de *backoff* [69].

A tecnologia *Ethernet* foi padronizada através da norma IEEE 802.3 [34], [70], na qual especifica-se a implementação da camada física através do emprego do mecanismo CSMA/CD.

Em relação ao modelo ISO/OSI, a *Ethernet* implementa a camada física e a de enlace de dados dividida em duas subcamadas, a de Controle de Acesso ao Meio (MAC – *Medium Access Control*) e a de Controle de Enlace Lógico (LLC – *Logical Link Control*), como mostrado na Figura 6.1.

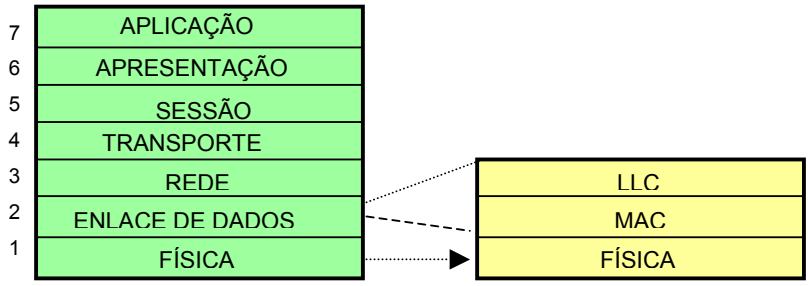


Figura 6.1. Camadas IEEE 802.3 em relação ao modelo ISO / OSI.

6.1.1 - Elementos de um Sistema *Ethernet*

Um sistema com base na tecnologia *Ethernet* consiste em três elementos básicos [69]:

a) Um conjunto de regras para acessar o meio;

b) Um quadro de *bits* padronizado, usado para transmitir dados através do sistema.

c) Meio Físico, usado para transmitir os sinais *Ethernet* entre computadores;

a) Controle de Acesso ao Meio

Através do controle de acesso ao meio, determinam-se os procedimentos que a rede de comunicação deve realizar para inserir dados no meio de transmissão. Este fato é especificado no padrão IEEE 802.3 (CSMA/CD).

A fim de esclarecer o funcionamento do protocolo CSMA/CD, suponha-se a seguinte situação: "um grupo de pessoas ao redor de uma mesa em uma sala escura. Cada pessoa componente do grupo esperará um tempo de silêncio antes de começar a falar (Detecção de Portadora). Acontece que nesse período todas as pessoas do grupo têm a mesma possibilidade de

falar (Acesso Múltiplo). No caso de duas pessoas começarem a falar ao mesmo tempo, o fato é detectado e a conversa se interrompe (Detecção de Colisão)" [69].

Em termos da tecnologia *Ethernet*, cada interface deve esperar até o canal ficar livre de sinais, assim, uma máquina pode iniciar a transmissão. Se uma estação estiver transmitindo existirá um sinal no canal de comunicação, denominada portadora. Todas as estações envolvidas devem esperar até o sinal de portadora acabar, para poder transmitir, este processo denomina-se detecção de portadora. Dado que todas as interfaces *Ethernet* possuem a mesma prioridade para transmitir, então, fala-se em acesso múltiplo. No caso de duas estações tentarem enviar dados ao mesmo tempo, haverá uma colisão de dados, este processo denomina-se detecção de colisão. Desta maneira, a transmissão é interrompida a fim de reenviar os dados.

b) Quadro *Ethernet*

O quadro *Ethernet* padronizado é mostrado na Figura 6.2.

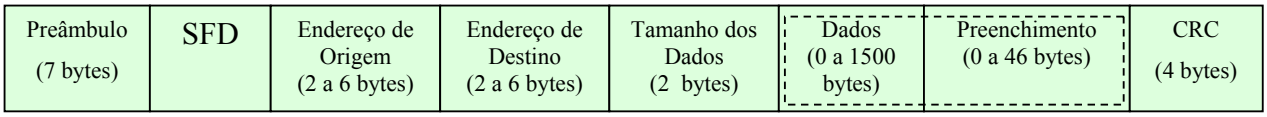


Figura 6.2. Quadro *Ethernet*.

b1) Preâmbulo: é uma seqüência de 56 *bits* utilizados para sincronismo, cada *byte* componente assume o valor "10101010".

b2) SFD (*Start Frame Delimiter*): delimitador de inicio de quadro. Indica o começo de um quadro, através de um *byte* que assume o valor "10101011".

b3) Endereços de Origem e Destino: o endereço da placa de rede de origem e destino, conhecido como endereço físico ou endereço MAC, do inglês *Médium Accesss Control*, especifica-se por meio de 48 *bits*. Os primeiros três *bytes* indicam o fabricante da placa de interface e, os três restantes, identificam a placa, sendo que o MAC de cada placa é único.

b4) Campo de Tamanho: faz parte da etapa de verificação, sendo que o campo é preenchido com dois *bytes*.

b5) Campo de dados: neste campo são colocados os dados propriamente ditos. O tamanho pode variar de 0 a 1500 *bytes*.

b6) Campo de Preenchimento: este campo é utilizado para garantir quadros mínimos de 64 *bytes*. A fim de esclarecer esta idéia será analisada a Figura 6.3, que ilustra as etapas envolvidas na detecção de uma colisão. No instante to, a estação A envia um quadro. Chamar-se-á de τ ao tempo que o quadro leva para chegar até a extremidade B da rede. Supondo que instantes antes do pacote atingir B, em τ-ε, B começa a transmitir, quando B detecta que está recebendo mais potência do que está produzindo, significa que uma colisão ocorreu, interrompe-se a transmissão e é gerada uma rajada de ruído de 48 *bits* para avisar a todas as estações.

Aproximadamente no tempo 2τ, o transmissor "vê" a saída e também interrompe a transmissão e, em seguida, ele aguarda um tempo aleatório antes de tentar novamente.

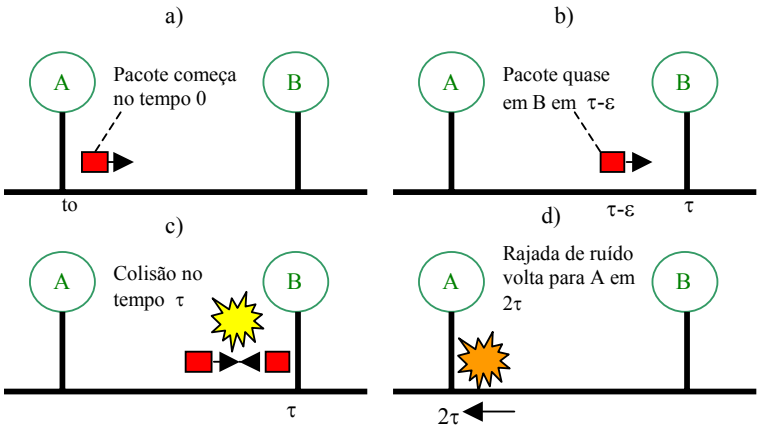


Figura 6.3. Detecção de uma colisão.

Restringindo o comprimento dos cabos a 2,5 Km e quatro repetidores entre duas estações (de acordo com a especificação IEEE 802.3) o tempo de ida e volta de um sinal no canal de comunicação pode ser limitado a 51,2 μs, o que a 10 Mbps corresponde a 512 *bits* ou 64 *bytes*, que é o tamanho de quadro mínimo.

Se uma estação tenta transmitir um quadro muito curto e uma colisão acontecer, a transmissão será concluída antes que uma rajada de ruído retorne em um tempo de 2τ. Assim, o emissor concluirá, erroneamente, que o quadro foi enviado com sucesso. Desta maneira, para evitar este problema, todos os quadros devem levar mais de 2τ para serem enviados, o que equivale a um quadro mínimo de 64 *bytes*. Caso o dado seja menor do que 46 *bytes*, o campo de preenchimento encarrega-se de fornecer a quantidade de *bits* necessária para montar um quadro válido.

b7) CRC: soma de verificação.

c) Meio Físico

No meio físico podem ser empregados diversos tipos de cabos, de acordo com a tecnologia *Ethernet* considerada. A seguir apresenta-se um resumo para as tecnologias mais utilizadas.

c1) *Ethernet* 10 Mbps

Esta tecnologia está disponível para os sistemas de cabeamento em banda básica[22] (10BASE) mencionados na Tabela 6.1.

Tabela 6.1. Cabeamento para *Ethernet* 10 Mbps.

Parâmetro	**10BASE 2**	**10BASE 5**	**10BASE T**	**10BASE F**
Tipo de Cabo	Coaxial fino	Coaxial grosso	Par trançado	Fibra óptica
Distâncias Máximas (m)	200	500	100	1000

c2) *Fast Ethernet* (100 Mbps)

Esta tecnologia está disponível para os sistemas de cabeamento em banda básica (100BASE) mencionados na Tabela 6.2.

Tabela 6.2. Cabeamento para *Fast Ethernet*.

Parâmetro	**100BASE T4**	**100BASE TX**	**100BASE FX**
Tipo de Cabo	Par trançado (4 pares)	Par trançado (2 pares)	Fibra óptica monomodo e multimodo
Distâncias Máximas (m)	100	100	2000

c3) *Gigabit Ethernet*

Esta tecnologia está disponível para os sistemas de cabeamento em banda básica, mencionados na Tabela 6.3.

[22] Banda Básica: corresponde à faixa de transmissão de sinais digitais não modulados.

Tabela 6.3. Cabeamento para *Gigabit Ethernet*.

Parâmetro	**1000BASE CX**	**1000BASE T**	**1000BASE LX**	**1000BASE SX**
Tipo de Cabo	Coaxial	Par trançado	Fibra óptica (monomodo)	Fibra óptica (multimodo)
Distâncias Máximas (m)	30	100	3000	550

6.1.2 – Desempenho da tecnologia *Ethernet*

O protocolo CSMA/CD não gera prioridades e, portanto, quando ocorre uma colisão, as estações envolvidas esperam um tempo aleatório e tentam a retransmissão. Sempre que ocorre colisão é necessário gerar um período de espera e, dessa maneira, o desempenho de redes *Ethernet* está ligado diretamente ao número de estações existentes na rede. Evidentemente, quanto mais máquinas, maior a probabilidade de ocorrerem colisões e, então, o desempenho da rede cai porque haverá períodos de espera. Entretanto, em redes com umas poucas estações a queda de desempenho não é tão relevante, já que as retransmissões ocorrem na escala de microsegundos, mesmo em redes *Ethernet* funcionando a 10 Mbps [35].

Existem inúmeros trabalhos sobre desempenho de redes baseadas em protocolos de acesso múltiplo, apresentando análises exaustivas baseadas em estudos probabilísticos. Porém, existem diferentes opiniões, o que acaba gerando certa confusão. O leitor pode recorrer às referências [71] e [72], onde são indicados trabalhos interessantes sobre desempenho da *Ethernet*.

A eficiência do canal de comunicação pode ser analisada, de maneira simplificada, através da seguinte equação [33]:

$$Ec = \frac{1}{1 + 2BLe / cF} \tag{6-1}$$

Sendo: B: Largura de Banda;
L: Comprimento do cabo;
e: *Slots*[23] de contenção por quadro;
c: Velocidade de propagação do sinal;
F: Comprimento de quadro.

[23]*Slot*: Cada um dos segmentos em que é dividido um espaço de comunicação e dentro dos quais uma mensagem pode ser armazenada. O número de *slots* é expresso através de um número inteiro.

Uma análise a partir da equação (6-1) permite concluir que, quando o segundo termo no denominador for elevado, a eficiência de rede será baixa. Em outras palavras, o aumento da largura de banda ou da distância da rede (produto BL), reduz a eficiência da rede para um determinado tamanho de quadro.

Na Figura 6.4 é quantificada a eficiência do canal em relação ao número de estações prontas, considerando um tempo $2\tau = 51{,}2\ \mu s$, na taxa de dados de 10 Mbps.

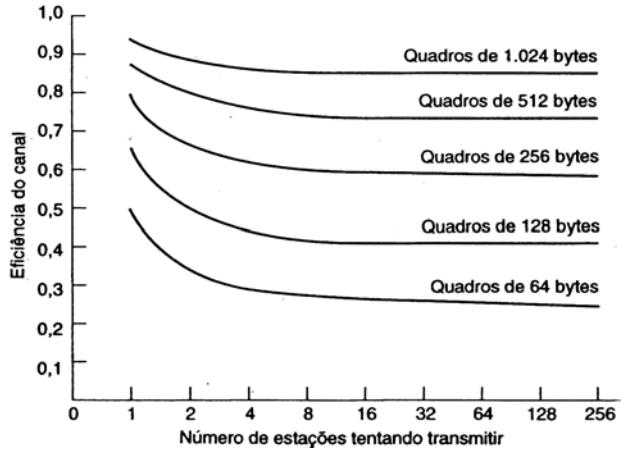


Figura 6.4. Desempenho da tecnologia IEEE 802.3.

Fonte: Tanenbaum, A.S. Redes de Computadores.

Na Figura 6.4 pode-se analisar que o aumento do número de estações tentando transmitir produz a diminuição da eficiência do canal, para um determinado tamanho de quadro. Conclui-se que existem dois problemas que influenciam no determinismo da *Ethernet*:

- A existência de colisões, que são necessárias pois fazem parte do mecanismo de funcionamento do protocolo CSMA/CD;
- A utilização do canal de comunicação, pelo fato do IEEE 802.3 não estabelecer prioridades.

6.1.3 - *Ethernet* na Automação Industrial

Basicamente a rede de campo[24] ideal deve possuir as seguintes características: determinismo, velocidade, volume de dados, flexibilidade e compatibilidade.

[24] De maneira geral, o termo rede de campo faz referência às tecnologias apresentadas no Capítulo 4.

A Ethernet-TCP/IP é a arquitetura de rede que possui as características necessárias para desempenhar a maioria das funções em diferentes níveis de um ambiente industrial, em virtude de sua capacidade de integração, por exemplo: supervisão, banco de dados, planejamento, etc. Porém, pelo fato de não ser determinista, os tempos de resposta podem ser insuficientes para algumas aplicações em nível de campo. Contudo, a *Ethernet* possui custo relativo baixo e capacidade para formar sistemas com tecnologias bem conhecidas e desenvolvidas. A *Ethernet* pode ser considerada uma rede de alta velocidade, basta analisar as velocidades das tecnologias de rede de campo apresentadas no Capítulo 4, para tirar essa conclusão. Estes fatos fazem com que a utilização da *Ethernet* como rede de controle nos sistemas DMC venha ganhando popularidade. É importante salientar também, que estas características são atrativas no âmbito da automação residencial e predial.

Os oponentes da utilização da *Ethernet* como rede de controle têm argumentado que ela nunca será determinista o suficiente para aplicações em tempo real. Porém, demonstrações de grandes companhias como a *Hewlett-Packard* têm provado que aplicações baseadas na *Ethernet* podem satisfazer tempos de resposta muito rigorosos [14]. No âmbito nacional, a *Ethernet*-TCP/IP tem sido testada em nível de rede de controle com resultados satisfatórios [73].

A arquitetura de rede para aplicações industriais tende a ser uma combinação da *Ethernet* com redes de campo já existentes e protocolos padronizados. Há alguns anos, a organização *Fieldbus Foundation* escolheu a *Ethernet* como especificação para a sua rede de controle, definindo uma rede chamada de H2, com velocidade em torno de 100 Mbps [14].

Voltando ao assunto do desempenho, à medida que mais e mais estações são conectadas a uma LAN baseada no padrão IEEE 802.3, o tráfego aumenta. Duas possíveis soluções seriam as seguintes [33]:

a) Aumento da velocidade: hoje são possíveis velocidades muito mais elevadas do que a original de 10 Mbps. A *Gigabit Ethernet* é uma tecnologia emergente e já definida no IEEE 802.3z. O aumento, por exemplo, de 10 para 100 Mbps melhora a eficiência da rede, porém, é preciso trocar as placas adaptadoras de 10 Mbps.

b) *Ethernet* Comutada: esta solução é baseada na inclusão de um comutador (*switch*) contendo um *backplane* de alta velocidade e espaço para, geralmente, 4 a 32 placas de linha plugáveis, cada uma contendo de 1 a 8 conectores. Com o emprego de um comutador, as mensagens ficam armazenadas provisoriamente na memória do microprocessador do comutador. Assim, é possível

realizar a transição de uma tecnologia baseada na estatística para uma tecnologia de comutação baseada em *hardware*, que possui um tempo de resposta previsível.

A principal desvantagem de um sistema comutado está associada ao custo do dispositivo comutador que possui um grau de sofisticação elevado se comparado com um *hub* tradicional. Os *hubs* são concentradores, responsáveis por centralizar a distribuição de quadros de dados, replicando, em todas suas portas, as informações recebidas pelas máquinas da rede. Ao contrário do *hub*, o comutador envia os quadros de dados somente para a porta de destino do quadro. Por outro lado, com o uso de comutadores, mais de uma comunicação pode ser estabelecida simultaneamente desde que as comunicações não envolvam portas de origem ou destino que já estejam sendo usadas em outra comunicação.

Hoje, fala-se em *Ethernet* industrial com diferenciação de tráfego, sustentada sobre a base da eliminação das colisões com o uso da *Ethernet* comutada e a previsão no enfileiramento com o emprego de mecanismos de qualidade de serviço especificados em IEEE 802.1q e IEEE 802.1d, os quais são fundamentados na diferenciação de tráfego e no escalonamento de prioridades.

Estudos nesta área permitem concluir que a *Ethernet* comutada em conjunto com o uso de mecanismos de diferenciação de tráfego apresenta-se como uma tecnologia bastante promissora para sistema de tempo real em ambientes de automação industrial [74].

Outro aspecto a ser considerado diz respeito aos atrasos e preparativos de gestão introduzidos pelo protocolo TCP/IP, pois o TCP é um protocolo orientado à conexão confiável, que garante a entrega de grandes blocos de dados, e isto tem um preço, justamente, a introdução de atrasos. Contudo, outro protocolo que pode ser utilizado na camada de transporte é o UDP, que foi projetado para ser mais rápido do que o TCP, entretanto, é um protocolo não orientado à conexão.

O conjunto de protocolos TCP/IP está se tornando um padrão para realizar as comunicações entre diferentes dispositivos, isso porque as redes baseadas nos protocolos TCP/IP têm o potencial necessário para satisfazer as novas necessidades de integração entre diferentes equipamentos que constituem uma rede. O TCP/ IP tem ainda algumas barreiras para se tornar um padrão, uma delas é a falta de determinismo. Porém, pelo fato do aumento da velocidade dos computadores e a crescente expansão da velocidade de comunicação das redes TCP/IP, não demorará muito tempo para que essas barreiras sejam superadas e o protocolo TCP/IP se torne um padrão de comunicação predominante em aplicações de instrumentação distribuída.

Conclui-se que o emprego da arquitetura *Ethernet*-TCP/IP na automação industrial pode não possuir determinismo para aplicações que precisem de um tempo de resposta rigoroso. No entanto, um investimento com a finalidade de implementar um ou vários comutadores pode melhorar em muito o desempenho do sistema.

Provavelmente, a característica mais importante da arquitetura *Ethernet*–TCP/IP seja sua capacidade de formar sistemas abertos e sua capacidade de integração com a Internet, o que motiva seu emprego no presente trabalho.

6.2 – Tecnologia de Lógica Programável

6.2.1 - Dispositivos Lógicos Programáveis

Os dispositivos lógicos programáveis (PLDs - *Programmable Logic Devices*) são circuitos integrados, configuráveis pelo usuário final e programáveis via *software*, utilizados para implementar funções lógicas envolvidas em sistemas digitais de complexidade variável.

A lógica programável abrange todos os tipos de circuitos lógicos configuráveis, incluindo dispositivos simples de baixa densidade, do tipo Matriz Lógica Programável (PAL - *Programmable Array Logic*), Matriz Lógica Genérica (GAL - *Generic Array Logic*), e dispositivos PLD de alta capacidade e elevado nível de integração como o PLD complexo (CPLD - *Complex Programmable Logic Device*) e a Matriz de Portas Programável em Campo (FPGA - *Field-Programmable Gate Array*). Na Figura 6.5, apresenta-se, de forma esquemática, uma classificação de PLDs, tecnologia CMOS, de propósito geral.

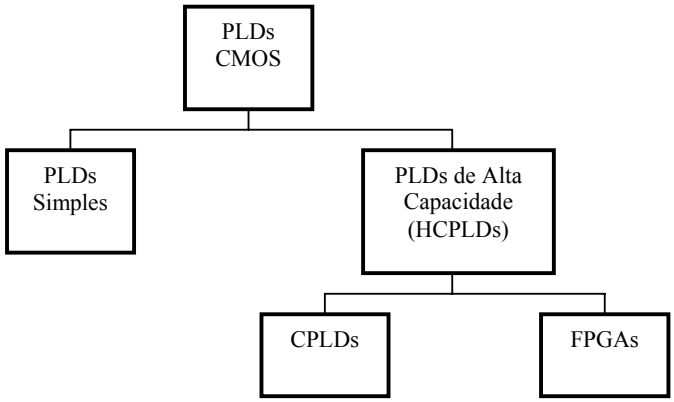


Figura 6.5. PLDs CMOS de propósito geral.

A arquitetura interna dos PLDs é composta de diversas estruturas repetidas, chamadas de células ou blocos lógicos. Cada célula é constituída por elementos de lógica combinacional junto a circuitos de lógica seqüencial. A fim de estabelecer a interface com o mundo exterior, têm-se blocos de entrada e saída, cuja função é conectar as células com os pinos do integrado. De modo a interligar os blocos internos, existem os canais e as linhas de interconexão. Tanto o sistema interno de interconexão, quanto o processo de fabricação, determinam as diferenças entre diferentes fabricantes de PLDs [75]. A titulo ilustrativo, na Figura 6.6, é mostrada a estrutura típica de uma FPGA.

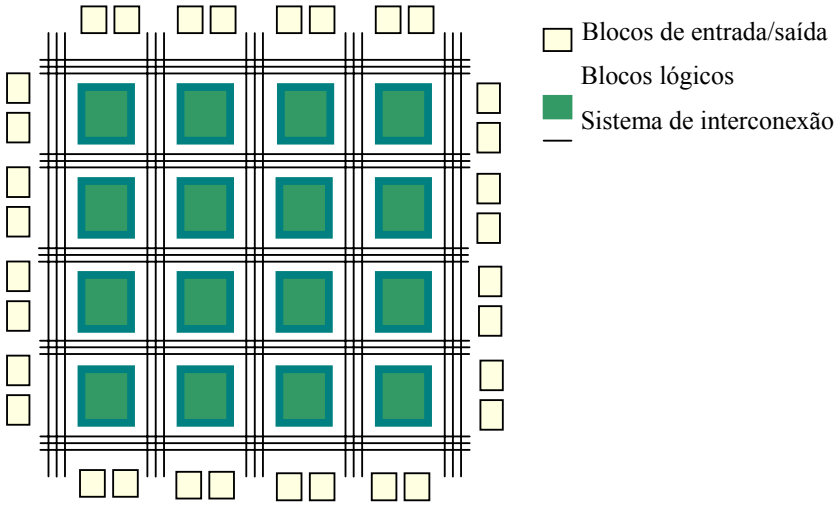


Figura 6.6. Estrutura típica de uma FPGA.

CPLDs e FPGAs diferenciam-se em suas estruturas internas de interconexão [76],[77]. Nas FPGAs, utilizam-se estruturas de conexões segmentadas, em que linhas de metal de comprimento variável são conectadas com as células lógicas, através de transistores de passagem. Por seu lado, os CPLDs, utilizam estruturas contínuas entre células, com linhas de metal de igual comprimento.

As estruturas de interconexão contínua eliminam a variabilidade de tempo associada com as estruturas de interconexão segmentada. Desta maneira, os retardos relacionados com as células lógicas são mais previsíveis em dispositivos CPLDs do que em FPGAs [76],[77]. Em geral, as células dos FPGAs são mais simples do que as dos CPLDs, entretanto, podem ser implementadas funções mais complexas utilizando várias das mesmas.

Os PLDs podem ser programados através de várias tecnologias disponíveis atualmente como a EPROM (*Erasable Programmable Read-Only Memory*), a EEPROM (*Electrically-Erasable Programmable Read-Only Memory*), a SRAM (*Static Random Access Memory*) e a FLASH[25]. Em geral, os dispositivos de maior capacidade utilizam a tecnologia RAM.

Quando comparados com outras opções de implementação de sistemas digitais como ASICs e ICs convencionais, os PLDs fornecem tempos de projeto e simulação menores, custo atrativo e, sobretudo, maior flexibilidade para realizar modificações futuras [76].

6.2.2 - Linguagens de Descrição de Hardware

Uma Linguagem de Descrição de Hardware (HDL - *Hardware Description Language*) é uma ferramenta empregada para descrever o comportamento e as estruturas de sistemas eletrônicos, através da utilização de uma estrutura textual, possibilitando a documentação, simulação, verificação e síntese de projetos de sistemas digitais de complexidade variável [78].

Através de uma HDL é possível descrever um sistema digital com um elevado nível de abstração, permitindo realizar os seguintes tipos de descrição:

a) Comportamental: descrição do quê o sistema deve fazer;

b) Funcional: descrição de como deve ser feito;

c) Estrutural: com quê deve ser feito.

Em geral, o emprego de uma HDL possibilita:

a) Especificar um sistema digital de modo geral;

b) Realizar uma boa documentação;

c) Utilizar hierarquias, desde subsistemas até componentes;

d) Utilizar funções parametrizáveis;

e) Empregar ferramentas de síntese.

a) A Linguagem VHDL

A VHDL (*VHSIC Hardware Description Language*) é uma linguagem padronizada, utilizada na descrição de comportamentos e componentes envolvidos em sistemas digitais. A VHDL tem produzido um enorme impacto nas metodologias de projeto de sistemas digitais, promovendo o

[25] *Flash*: Tecnologia de memória baseada na EEPROM.

projeto tipo *top-down*[26], de forma semelhante aos programas desenvolvidos em linguagens de programação de alto nível como Pascal ou C++. Tem contribuído para o estabelecimento de uma nova metodologia de projeto, fazendo com que um sistema digital possa ser descrito com um elevado grau de abstração, mais distante dos detalhes de baixo nível como transistores e portas lógicas [78],[79].

A linguagem VHDL foi padronizada, a princípio, através do padrão IEEE 1076-1987 [80], e posteriormente, em 1993, foi atualizada através do padrão IEEE 1076-1993 [81]. Devido a este fato, a VHDL é conveniente para ser empregada em sistemas abertos, possibilitando independência tecnológica e reutilização de códigos.

b) Estrutura de uma Descrição em VHDL

Uma descrição em VHDL é composta fundamentalmente por dois blocos, a entidade e a arquitetura. A entidade é o bloco elementar de toda descrição VHDL. Neste bloco são definidas as entradas e as saídas do elemento a ser criado. A arquitetura descreve o comportamento da entidade. Toda descrição deve conter uma arquitetura para poder ser simulada [82]. Na Figura 6.7 mostra-se a descrição comportamental de uma porta lógica AND.

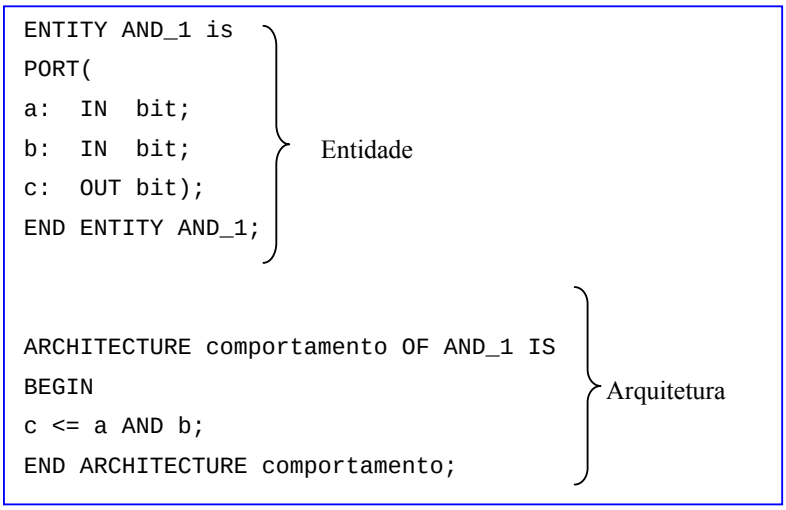


Figura 6.7. Descrição VHDL comportamental de uma porta AND.

[26] *Top-down* é uma metodologia de projeto que permite conceber um sistema através de sua especificação funcional, em contraste com a metodologia *bottom-up*, em que o projetista constrói o sistema a partir dos componentes elementares.

6.2.3 - Ambientes de Síntese

Junto à utilização de uma HDL é necessário o emprego de um ambiente de desenvolvimento para projeto de sistemas digitais. Este ambiente é propiciado por um *software* que geralmente é fornecido pelas mesmas empresas fabricantes de PLDs. Desta maneira, é possível realizar a descrição em VHDL de um determinado projeto, que, logo depois, o *software* se encarrega de sintetizar. Assim, o sistema pode ser simulado e, finalmente implementado, com um dispositivo PLD. Para tal fim, os ambientes de desenvolvimento fornecem o modo de programação no sistema (ISP – *In System Programmability*). Isto é, a capacidade de programar o dispositivo enquanto se encontra na mesma placa em que opera. Em geral, os ambientes fornecem os recursos da Interface JTAG (*Joint Test Action Group*) padronizada através do IEEE 1149.1. Dentre outras coisas, através da JTAG é possível programar um PLD no modo ISP, conectando, através de um cabo especial, a porta paralela do PC em que roda o ambiente de síntese, com o conector JTAG da placa que contém o PLD.

O processo de síntese de um projeto digital descrito através de uma HDL passa por diversas etapas, como mostrado na Figura 6.8.

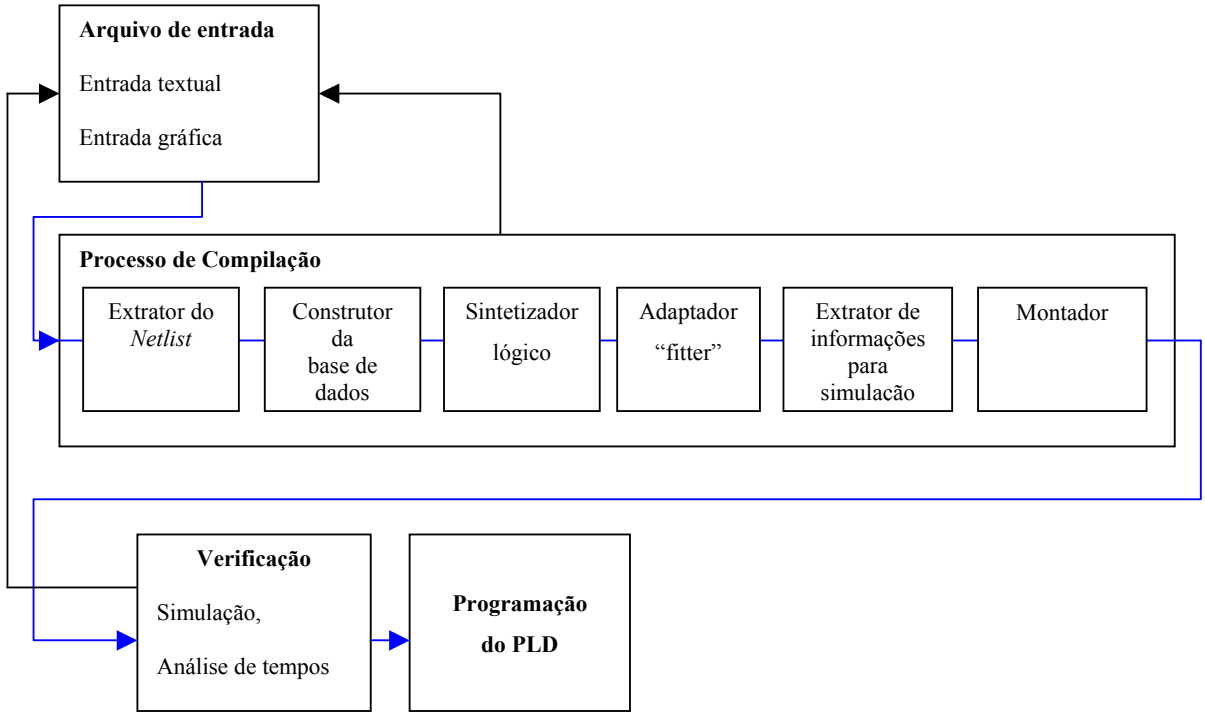


Figura 6.8. Processo de síntese de um projeto digital descrito em HDL.

a) Arquivos de Entrada

O arquivo de entrada pode ser textual ou gráfico. Por exemplo, o projeto textual da Figura 6.8, que é salvo como AND_1.vhd. Entretanto é possível criar um projeto hierárquico composto por diversos blocos funcionais descritos em VHDL. Assim é possível colocar os símbolos dos blocos criados, em um editor gráfico e, dessa maneira, ter um arquivo de entrada gráfico.

b) Processo de Compilação

Este processo realiza a minimização e a síntese lógica, efetua o processo de adequação do projeto no PLD escolhido e gera os dados necessários à implementação do projeto no dispositivo. O processo de compilação passa pelas seguintes etapas [83], [84]:

b1) Extrator do *Netlist*[27]: converte cada arquivo do projeto em arquivo de *netlist*, criando os arquivos necessários à determinação da hierarquia. A hierarquia criada pode ser visualizada no ambiente;

b2) Construtor da base de dados: nesta etapa cria-se uma base de dados relacionada com o projeto. O construtor examina a consistência do projeto e testa a conectividade dos nós envolvidos no sistema na procura de erros;

b3) Sintetizador lógico: realiza o processo de minimização e síntese lógica;

b4) Adaptador: responsável por adequar as necessidades do projeto no PLD escolhido. Nesta etapa é gerado o arquivo de relatório do projeto, o qual constitui um recurso de grande utilidade, pois nele constam informações como o número de células utilizadas, memória empregada, quantidade de entradas e saídas usadas, etc;

b5) Extrator de informações para simulação: extrai as informações necessárias para simular o projeto em um diagrama de tempos.

b6) Montador: cria uma imagem dos pinos, células lógicas e assinaturas feitas pelo adaptador na forma de arquivos de saída para programar um dispositivo PLD. Por exemplo, AND_1.pof (*programmer object files*), para programar um dispositivo com base na tecnologia EPROM e AND_1.sof (*SRAM object files*), para programar um dispositivo com base na tecnologia SRAM.

Cumpre salientar que, no caso de acontecer algum erro durante a compilação, deve-se retornar ao arquivo de entrada e corrigir o problema relatado.

---

[27] *Netlist*: lista que contém as informações sobre a conectividade entre os componentes dentro de um projeto.

c) Verificação

A ferramenta fundamental que possibilita a verificação do projeto é o simulador. Através deste recurso é possível verificar se o sistema satisfaz os requisitos de projeto. Se for necessário, deve-se retornar ao arquivo de entrada para modificá-lo.

d) Programação

Finalmente, o projeto deve ser programado em um PLD cuja capacidade, isto é, número de células, pinos de entrada/saída e memória, satisfaça às especificações do projeto.

De modo ilustrativo, na Figura 6.9 é mostrada a AND_1, criada através da entrada textual da Figura 6.7 (AND_1.vhd). Pode-se observar o símbolo criado (AND_1.sym), a hierarquia, parte do relatório (AND_1.rpt) e a simulação correspondente (AND_1.scf), no ambiente de síntese *Max+Plus® II* da empresa *Altera*.

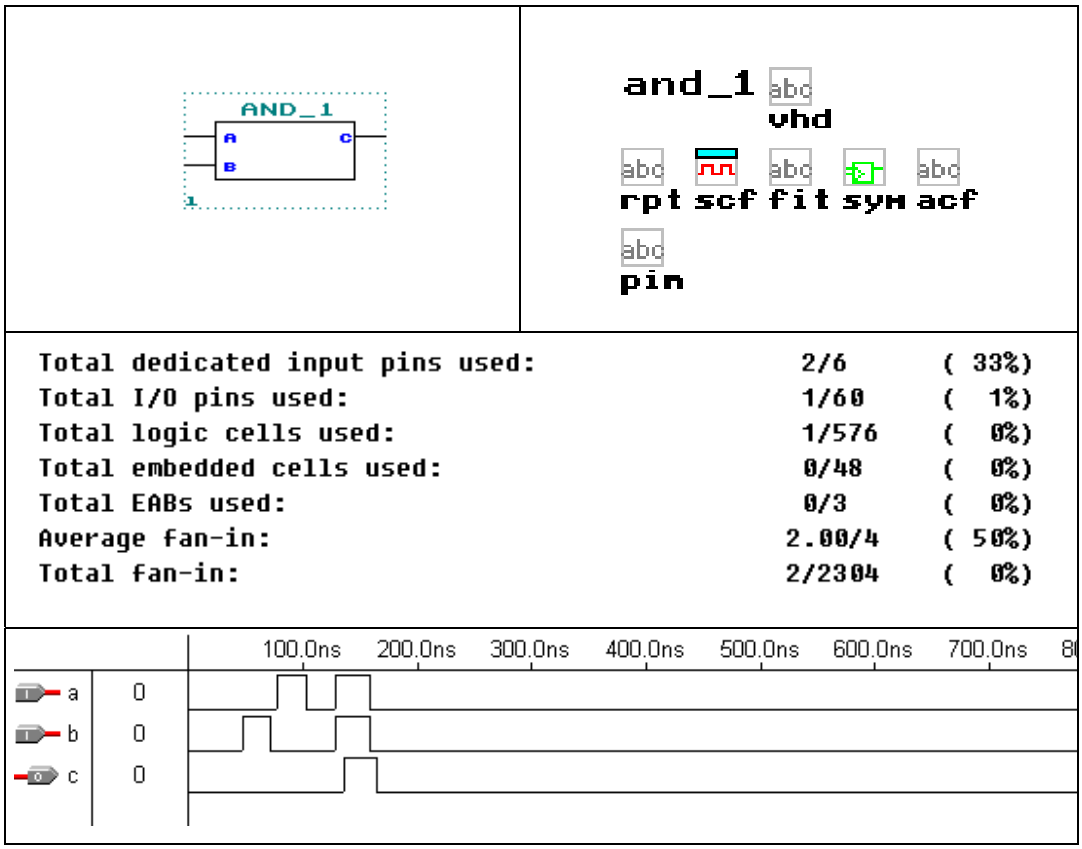


Figura 6.9. Exemplo de arquivos criados no ambiente *Max+Plus®II*.

6.2.4 - Desempenho de PLDs

Algumas das vantagens de se empregar dispositivos lógicos programáveis foram mencionadas na seção 6.2.1. Analisar-se-ão, a seguir, algumas possíveis limitações destes dispositivos.

Uma possível deficiência dos PLDs está associada à quantidade de memória interna disponível; sistemas de grande complexidade devem ser implementados com uma memória externa, perdendo as vantagens inerentes de um sistema "*on chip*". Isto é verdade ao considerar PLDs antigos, pois hoje existem dispositivos *multicore* e de alta capacidade que podem suprir essas deficiências, embora o custo seja maior se comparado com o custo dos dispositivos tradicionais.

Outro aspecto a ser levado em conta é que os PLDs podem ser empregados para projetar microprocessadores e circuitos associados. Por exemplo, quando se faz processamento baseado em FPGA, o acesso a um elevado número de registradores internos requer um projeto cuidadoso por parte do projetista, porque o sistema desenvolvido pode não estar otimizado.

Em relação ao projeto de sistemas digitais, a velocidade é um ponto importante. Em termos de freqüência, o desempenho de um projeto sintetizado em um determinado PLD, depende, por exemplo, de como é feito o roteamento interno dentro da estrutura matricial de uma FPGA. Não existe apenas uma possibilidade, o fato da compilação de um projeto ter sido realizada com sucesso, não significa que tenha sido feita da maneira ótima.

O ambiente *Max+Plus® II* fornece uma ferramenta denominada *Clique* que pode ser usada para o roteamento lógico de uma forma otimizada. Esta ferramenta pode ser utilizada em conjunto com o analisador de tempos, a fim de quantificar o desempenho de um projeto em termos de freqüência de operação, como será analisado na seção 8.4 do Capítulo 8.

Finalizando, as mencionadas, são algumas das considerações a serem levadas em conta quanto ao desempenho de PLDs. Cumpre salientar que os aspectos mencionados dependem do tipo de projeto a ser desenvolvido e a sua possível aplicação. Em alguns casos o fator determinante de projeto será a memória disponível e em outros será velocidade, ou ambas.

Ao projetar um STIM são necessários em torno de 100 *bytes* por cada bloco TEDS implementado. Para uma aplicação com poucos canais transdutores, digamos da ordem de 20, o fator relativo a recursos de memória não será determinante, entretanto, pode sê-lo a velocidade de transporte de dados sobre a TII de acordo com o tipo de aplicação.

6.3 – Tecnologia Java

6.3.1 - Introdução

Java nasceu no ano de 1990 da mão de *James Gosling*, da empresa *Sun Microsystems*, para aplicações relacionadas com dispositivos eletrônicos de consumo. Inicialmente, a idéia era a de criar uma linguagem baseada nas existentes C e C++, que pudesse ser facilmente adaptada a qualquer ambiente de execução [85]. Assim, nascia a linguagem *Oak*, que mais tarde passou a ser denominada Java, pelo fato de ter-se conferido a existência de uma outra linguagem com esse nome. A *Oak* foi utilizada no *Green Project*, financiado pela *Sun Microsystems*, na construção do sistema *Star Seven*, que possibilitou o controle de vários aparelhos eletrônicos de uma casa através do toque em uma tela e de um sistema de televisão interativa [86]. Contudo, os projetos não chegaram a se converter em produtos finais. Impulsionada pela Internet, Java foi adaptada e lançada oficialmente, pela *Sun*, no ano de 1995, passando a ser incorporada nos navegadores mais comuns.

Java é uma linguagem puramente orientada a objeto que constitui uma coleção de Interfaces de Programação Aplicativos (APIs – *Application Programming Interfaces*) para o desenvolvimento de diversas aplicações em várias áreas da engenharia. Java é um ambiente de execução presente em navegadores, celulares, sistemas embarcados, eletrodomésticos, etc. Desta maneira, Java não deve ser considerada apenas como uma linguagem, mas como uma tecnologia.

Em termos computacionais, a programação orientada a objetos é uma metodologia de implementação, na qual os programas são organizados como sendo uma coleção de objetos que cooperam entre si, sendo que cada um desses objetos representa uma instância de uma classe. As classes fazem parte de uma hierarquia e se vinculam através de relações de herança [87]. A programação orientada a objetos sustenta-se sobre a base de um conjunto de condições comportamentais. Um desses elementos, tal vez o mais importante, denomina-se abstração.

Através de uma abstração é possível definir as características essenciais de um objeto que pertence ao mundo real, isto é, seus atributos e seu comportamento, a fim de modelá-lo, logo depois, em um objeto de *software*. O principal recurso para suportar a abstração é a classe. Uma classe descreve um grupo de objetos que compartilham características semelhantes. Estas características especificam-se por meio dos atributos e comportamentos. Em termos da

programação orientada a objetos, diz-se que um objeto constitui uma instância de uma classe determinada. Enquanto um objeto é uma entidade concreta que existe no tempo e espaço, uma classe representa apenas uma abstração. Por exemplo, a classe transdutor terá alguns atributos, tais como: tipo de transdutor, grandeza física envolvida e identificação em um sistema. As operações que podem ser definidas para essa classe são, detectar, executar ação, etc. Desta maneira um objeto poderia ser um sensor de temperatura com determinada identificação, detectando temperatura ambiente. Assim sendo, um sensor de temperatura e um sensor de pressão são objetos da mesma classe, porém, possuem atributos diferentes.

6.3.2 - Funcionamento da Linguagem Java

A tecnologia Java introduz o conceito de Máquina Virtual Java (JVM - *Java Virtual Machine*). Depois de compilado, um programa em Java se transforma em *byte-codes*. Atualmente os *byte-codes* não podem ser executados em nenhuma arquitetura sem a utilização de uma ferramenta específica para tal fim. Entretanto, podem ser interpretados. O interpretador Java, é a implementação da JVM voltada para uma determinada plataforma. Desta maneira, a diferença de outras linguagens, Java é compilada e interpretada, introduzindo portabilidade.

Quanto à compilação e execução de um programa em Java é interessante analisar um exemplo, tomando um programa genérico que será chamado de *Transdutor*. As etapas necessárias à execução do programa são as seguintes:

a) Instalar o ambiente de desenvolvimento, por exemplo, o Kit de Desenvolvimento Java (JDK – *Java Development Kit*) da *Sun Microsystems*, que possui várias versões [88]. Para a instalação recorre-se ao Apêndice D.

b) Escrever o programa em um editor de textos e salvá-lo com a extensão .java, dentro da pasta jdk...\diretório. Exemplo: Transdutor.java no caminho C:\ cd j2sdk1.4.0\testes;

c) Editar o *prompt* (DOS) ou *shell* (*Linux*) para compilar e executar o programa;

d) Compilar o programa. C:\ cd j2sdk1.4.0\testes> javac Transdutor.java;

e) Executar o programa. C:\ cd j2sdk1.4.0\testes> java Transdutor.

Durante o processo de compilação é gerado o arquivo *Transdutor.class*, o qual será interpretado pela JVM.

Um programa em Java é composto por métodos, através dos quais são executadas as operações. Visando esclarecer o conceito será estudado o exemplo apresentado na Figura 6.10a,

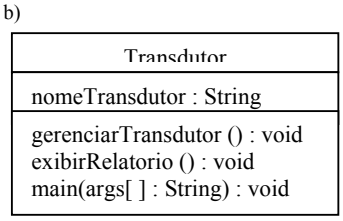
onde é possível analisar como é criado um objeto e como é realizado o processo de chamada de métodos. Se bem que, o exemplo é extremamente simples, pois, apenas imprime uma palavra na tela do monitor, ele tem como objetivo ilustrar como o processo pode ser feito de uma maneira organizada e modular. Na Figura 6.10b, ilustra-se o diagrama de classe correspondente e, na Figura 6.10c, a mensagem que aparece na tela do monitor.

A classe que representa a *Transdutor* possui um atributo denominado *nomeTransdutor* e seu modo de acesso é *private*, isto é, apenas os métodos da classe *Transdutor* conseguem acessar o atributo. Logo a seguir, descreve-se o método principal (*main*), onde se inicia a execução do programa. Cria-se, então, uma variável do tipo *Transdutor*, denominada *sensor1* e, logo depois, um novo objeto da classe *Transdutor*. Finalmente, é invocado o método *gerenciarTransdutor ()* do objeto *sensor1*. No corpo do programa, tem-se o método *gerenciarTransdutor ()* que dá um valor para o atributo e chama o método *exibirRelatorio ()*. Este método exibe o relatório na saída padrão do sistema, ou seja, na tela do monitor. O relatório, neste exemplo, resume-se simplesmente ao nome do transdutor.

a)

```
public class Transdutor {
        // Declaração dos atributos da classe
        private String nomeTransdutor;
        // Método Principal
        public static void main( String args[] )
        {       //Criação de um novo objeto da classe transdutor
                Transdutor sensor1;
                sensor1 = new Transdutor();
                sensor1.gerenciarTransdutor(); }
       // Declaração dos métodos da classe
        public void gerenciarTransdutor()
        {       nomeTransdutor = "Sensor de Temperatura_1";
                exibirRelatorio(); }
        public void exibirRelatorio()
        {       System.out.println( nomeTransdutor ); }
}
```

b)



c)

```
C:\>cd j2sdk1.4.0\testes

C:\j2sdk1.4.0\testes>javac Transdutor.java

C:\j2sdk1.4.0\testes>java Transdutor
Sensor de Temperatura_1
```

Figura 6.10. a) Exemplo em Java; b) Diagrama de classe e c) Exibição na tela do monitor.

6.3.3 - Principais Características da Linguagem Java

Algumas das principais características da linguagem Java são as seguintes:
a) Orientada a Objeto: Java é puramente orientada a objeto, possibilitando que uma classe herde o comportamento de sua superclasse;
b) Robusta: em Java os *byte-codes* são interpretados antes da execução do programa e possui coleta automática de lixo (g*arbage collector*) evitando erros de gerenciamento de memória;
c) Portável: a linguagem Java foi projetada para ser executada em qualquer sistema que implemente a Maquina Virtual Java;
d) Suporta aplicações em ambientes distribuídos;

6.3.4 – API RMI

O pacote para Chamada de Métodos Remotos (RMI – *Remote Method Invocation*) pertencente à tecnologia Java, fornece um mecanismo que possibilita a chamada de um método que pertence a um determinado objeto, sendo que este objeto pode existir no contexto da mesma máquina, ou de uma diferente. O RMI é em essência, um mecanismo orientado a objeto para Chamada de Procedimentos Remotos (RPC – *Remote Procedure Calls*).

Existem três elementos bem definidos para suportar a chamada remota de métodos [89]:
a) O Cliente, responsável por chamar um método em um objeto remoto;
b) O Servidor, que é o elemento ao qual pertence o mencionado objeto;
c) O Registrador de Objetos, que é um servidor de nomes, o qual relaciona objetos com nomes.

a) Arquitetura RMI

Na Figura 6.11 é apresentado o diagrama da arquitetura RMI. A *Skeleton* e a *Stub* são classes geradas ao se compilar o programa do servidor no entorno RMI, por meio do comando *rmic*. Ambos elementos RMI se comunicam através de uma camada de referência remota e entre eles estabelecem-se os processos de *marshaling*[28] e *demarshaling* da informação transmitida.

[28]*Marshaling*: processo de conversão de um formato criado com determinada linguagem, em um formato de transmissão (*on-the-wire*). *Demarshaling*: processo inverso ao anterior.

A transmissão dos dados, via rede, se dá através do protocolo TCP/IP, na camada de transporte, de maneira transparente.

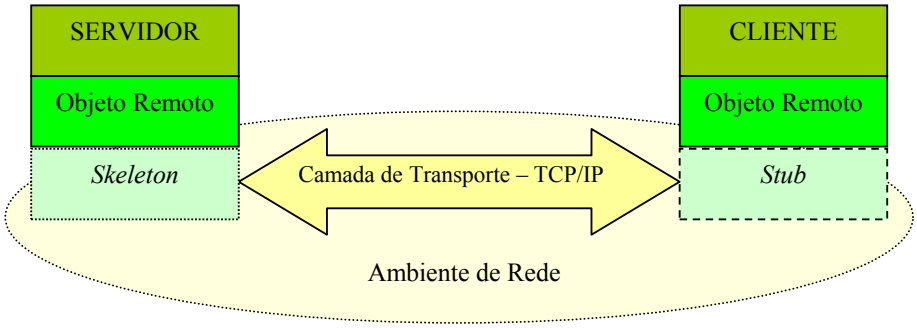


Figura 6.11. Diagrama da arquitetura RMI.

É importante esclarecer que os objetos remotos não viajam de uma máquina para outra, pois as classes geradas encarregam-se do processo de transmissão da informação. Do lado cliente, a camada *stub* fica responsável por aceitar as chamadas remotas e delegá-las, através da rede, à camada *skeleton* do lado servidor. A *skeleton* recebe as chamadas realizadas, via rede, e as entrega para o objeto remoto requisitado.

b) Exemplo RMI

Em uma aplicação RMI com um lado cliente e um lado servidor, deve existir uma interface. Em Java, uma interface fornece um espaço para declarar um método. Desta maneira, uma ou várias classes podem implementar o comportamento definido na interface, resolvendo-se, assim, o problema da linguagem Java não possuir herança múltipla.

Em uma aplicação distribuída, as implementações residem em máquinas virtuais diferentes. Deste modo, os objetos que possuem métodos que podem ser chamados por máquinas diferentes são denominados objetos remotos. Cumpre salientar que um objeto converte-se em remoto ao implementar uma interface remota e, para isso, é necessário que a interface herde o comportamento da classe *Remote* do pacote RMI. Além disso, cada método definido na interface deve declarar uma exceção remota (*RemoteException*), além de qualquer outra exceção adicional.

A seguir será analisado um exemplo simples com a finalidade de ilustrar como implementar um servidor, um cliente e a interface. Na Figura 6.12 mostra-se o código de uma interface simples, denominada Teste.

```
/*Interface*/
import java.rmi.*;

//Teste estende a classe Remote
public interface Teste extends Remote{
        public String IdSensor(String dado) throws RemoteException;
}
```

Figura 6.12. Exemplo de interface em Java.

O programa do servidor implementa a interface Teste e herda o comportamento da classe *UnicastRemoteObject*, pertencente à API RMI. No método principal deve ser criado e registrado um novo objeto remoto. O método *rebind*, da classe *Naming*, relacionará o objeto remoto com nome que tem sido especificado. Na Figura 6.13 pode-se analisar o código fonte do servidor.

```
/* Este programa implementa o Servidor */
import java.rmi.*;
import java.rmi.server.*;
/* A classe Servidor estende a classe UnicastRemoteObject e implementa a interface Teste */
public class Servidor extends UnicastRemoteObject implements Teste {
       //Declaração de um método Construtor
       public Servidor() throws RemoteException {
       }
       public String IdSensor (String dado) throws RemoteException {
           System.out.println("Codigo de identificacao enviado pelo cliente: " + dado);
           return dado;
       }
       //Método principal
       public static void main(String[] args) {
           try {//tentativa
           String nome = "//localhost/RecebeDado";
           //Criar um novo objeto remoto
           Teste teste = new Servidor();
           //Relacionar nome com objeto
           Naming.rebind(nome, teste);
           System.out.println("\n" + " Servidor no Ar" + "\n" + "--------------");
               } catch (Exception problemas) {//Tentou e não conseguiu?
           System.err.println("exception: " + problemas);
           }
         }
}
```

Figura 6.13. Exemplo de implementação de um servidor na linguagem Java.

No exemplo da Figura 6.14 mostra-se o código para a implementação do programa cliente. Através de uma interface gráfica, um cliente entra com o código de identificação de um sensor, que é declarada como uma *string* para ser interpretada pelo programa. O método *lookup*, da classe *Naming*, solicita e chama a referência do objeto remoto. Logo depois, o valor introduzido pelo cliente é passado como parâmetro para interface .

```
/*Este programa implementa o Cliente*/
import java.rmi.*;
import java.io.*;
import javax.swing.JOptionPane;

public class Cliente {
        static public String getNome() {
        //Entrar com uma String através de uma interface gráfica simples
          String id = JOptionPane.showInputDialog
         ("Entre com o codigo de identificacao do sensor");
          return id;
          }


        //Método Principal
        public static void main (String[] args) {
          //Utilizar o próprio host
          String host = "localhost";
          try {//tentativa
          Teste adq = (Teste)
          //Solicitar a referência de um objeto remoto e invocá-la
          Naming.lookup("rmi://localhost/RecebeDado");
          String id_sensor;
         //Obter código e passá-lo como parâmetro para a Interface
          id_sensor = getNome();
          adq.IdSensor(id_sensor);
          System.out.println("O Dado foi Enviado com Sucesso");
            }
          catch (Exception remota) {//Tentou e não conseguiu? => Mostre a exceção
           System.err.println("Erro na Conexao Remota: " + remota.getMessage());
          remota.printStackTrace();
          System.exit(0);  }
       }
}
```

Figura 6.14. Exemplo de implementação de um cliente na linguagem Java.

Após os programas terem sido escritos, devem ser compilados: o programa que implementa o servidor, o cliente e a interface. Na seqüência deve-se iniciar o registro RMI e compilar-se o programa servidor no ambiente RMI, empregando o comando *rmic*. Depois dessas etapas, o servidor pode ser executado a fim de colocá-lo no ar, como mostrado na Figura 6.15a. Uma vez no ar, o servidor esperará a petição de um serviço por parte de um cliente e, para isso, deve ser executado o programa cliente, como mostrado na Figura 6.15b.

Neste exemplo, ao executar o programa cliente, aparece uma interface gráfica por meio da qual solicita-se a entrada de um código de identificação de sensor. O valor ingressado será enviado para o servidor. Embora o exemplo apresentado seja básico, através dele é possível notar que, uma vez no servidor, um valor pode passar a fazer parte de uma base de dados, ou pode ser um comando remoto ou um endereço de função IEEE 1451.2.

Figura 6.15. Compilação e execução dos programas: a) servidor, b) cliente.

6.3.5 - API de Comunicações

O pacote de comunicações (API Comm) é uma extensão do *kit* JDK e, portanto, deve ser instalado à parte[29]. Através do pacote *javax.comm* é possível estabelecer comunicação com a porta série RS-232 e com a porta paralela IEEE-1284 de um Computador Pessoal.

[29] O pacote *comm* pode ser facilmente obtido no *site* da *Sun Microsystems* [88].

A grande vantagem da *comm* é que permite realizar este tipo de atividades totalmente na linguagem Java, não havendo a necessidade de serem empregados métodos nativos, como é o caso do JNI (*Java Native Interface*). Como se sabe, através do JNI, a comunicação com as portas, em baixo nível, pode ser implementada em C, C++ ou *assembly* e, o processamento, em alto nível, em Java. Entretanto, perde-se a portabilidade.

No entanto, o pacote c*omm* é ainda uma ferramenta nova e nota-se uma falta de amadurecimento, o que torna difícil sua aplicação.

Na Figura 6.16 é mostrado um exemplo básico de utilização do pacote *comm*, no caso, o programa *Portas*, identifica a porta paralela e informa se está ocupada ou disponível. Na Figura 6.17 é mostrada a exibição do resultado na tela do monitor.

```
/* Este programa identifica a porta paralela e informa se está ocupada ou disponível */
import java.io.*;
import java.util.*;
import javax.comm.*; //importar o pacote comm
class Portas {
   //novos atributos: portas e portaId
   static Enumeration portas; //Enumeration, classe do pacote io
   static CommPortIdentifier portaId; //CommPortIdentifier, classe do pacote comm

   public static void main(String[] args) {
     //Identificar portas
     portas = CommPortIdentifier.getPortIdentifiers();
     procurarLPT1(); }

     public static void procurarLPT1() {
     //Procurar todas as portas
     while (portas.hasMoreElements()) {
       portaId = (CommPortIdentifier) portas.nextElement();
       //Caso porta = LPT1, exibir mensagem
       if (portaId.getName().equals("LPT1")) {
         System.out.println("\n"+ "\n"+ "Porta LPT1 encontrada" + "\n");  }
     }
     //Caso a porta estiver disponível, exibir mensagem "Disponível", caso
     //contrário, exibir qual a aplicação usuária
     if (portaId.isCurrentlyOwned()) {
     System.out.println("OCUPADO por: " +  portaId.getCurrentOwner());
     } else  {
     System.out.println("DISPONIVEL" + "\n" + "\n"); }
    }
 }
```

Figura 6.16. API Java de Comunicações; Exemplo básico de aplicação.

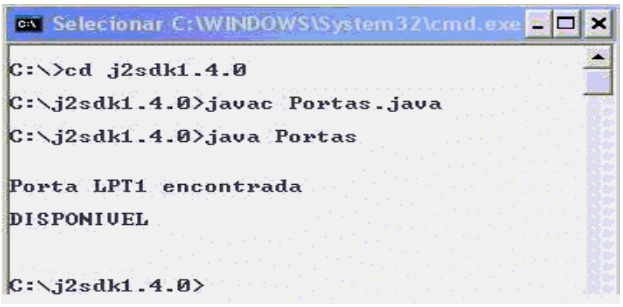


Figura 6.17. Exemplo básico de Java comm; Exibição na tela do monitor.

6.3.6 - Códigos Objeto e Java

A tecnologia Java permite que um código objeto possa ser carregado dentro de um programa escrito em Java através do emprego de JNI, como mencionado na seção anterior. A expressão código objeto faz referência a códigos escritos em C/C++ ou na linguagem *assembly.*

O emprego dessa solução pode ser motivado pela dificuldade na utilização do Java *comm* para implementar operações de leitura através da porta paralela ou, fundamentalmente, pelo aumento da velocidade das comunicações pelo fato do código objeto rodar mais rápido do que o código escrito em Java.

As etapas necessárias para chamar um método nativo através de um aplicativo desenvolvido em Java são as seguintes [90]:

a) Escrever um código em Java que chame um método nativo, por exemplo: Teste.java;

b) Compilar o código escrito em Java;

c) Criar o arquivo de cabeçalho Teste .h, usando o utilitário *javah*. O arquivo de cabeçalho gera um protótipo de função para o método nativo definido na classe Teste.java;

d) Escrever a implementação do método nativo que execute a atividade desejada, usando código objeto. Por exemplo: Teste.c;

e) Compilar o código escrito em C.

f) Criar uma biblioteca de ligação dinâmica (dll – *dynamic link library*) a ser carregada através da classe Teste.java;

g) Executar o programa Teste.

A desvantagem desta solução é o comprometimento da portabilidade, porque a interpretação do arquivo de cabeçalho depende do sistema operacional utilizado. O código nativo pode precisar

ser compilado com o mesmo compilador C que é utilizado pelo ambiente de execução Java (JRE - *Java Run-Time Environment*). Na plataforma Win32, por exemplo, será necessário o Visual C++ versão 4 ou superior.

Em termos práticos, o emprego de JNI não é tão simples, fundamentalmente no que diz respeito à criação da biblioteca de ligação dinâmica, o que pode precisar de certa experiência como programador por parte do projetista. Afortunadamente existe uma solução que permite a comunicação com a porta paralela de um PC para realizar operações de leitura e que tem como objetivo direcionar os problemas comentados e as deficiências da API de comunicações. Esta solução é o pacote de domínio público *Parport* desenvolvido por *J.C. Del Cid Portillo* [91].

a) O Pacote *Parport*

Atualmente, *Parport* é um pacote que pode ser instalado tanto em sistemas operacionais *Windows* quanto *Linux* e possibilita a comunicação com a porta paralela de um PC convencional utilizando Java. Entretanto, *Parport* emprega a linguagem C para realizar comunicação direta com a porta paralela. A vantagem desta solução é que o projetista não precisa programar em C, pois este aspecto já vem resolvido pelo próprio pacote.

As etapas necessárias à instalação do *Parport* estão no Apêndice D. Após o pacote ter sido corretamente instalado é possível criar uma classe em Java que importe o pacote *Parport*, a fim de escrever e ler dados usando a porta paralela, como ilustrado no exemplo da Figura 6.18.

```
import parport.ParallelPort; //importar a classe ParallelPort do pacote Parport
class IO {
   public static void main ( String []args ) {
      //criar um objeto da classe ParrallelPort, 0x378 é o endereço base para a porta LPT1
      ParallelPort lpt1 = new ParallelPort(0x378);
      int aByte;
      //Ler um Byte a partir do registrador de estados da porta paralela
      aByte = lpt1.read();
      System.out.println("VALOR LIDO: " + aByte);
      //Escrever o valor 85 no registrador de dados da porta paralela
      aByte = 85;
      lpt1.write(aByte);
      System.out.println("VALOR ESCRITO: " + aByte);
           }
      }
```

Figura 6.18. Leitura e escrita utilizando a porta paralela.

Na Figura 6.19 observa-se o resultado da execução do programa IO.java na tela do monitor, salientando-se que a compilação e a execução do programa se realiza da forma tradicional.

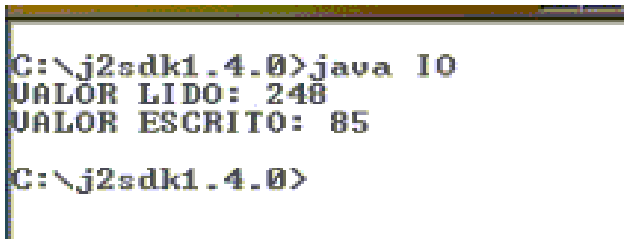


Figura 6.19. Exemplo de leitura e escrita utilizando a porta paralela; Exibição na tela do monitor.

b) O Aplicativo *Userport*

Por motivos de segurança, os sistemas operacionais *Windows® NT*, *2000* e *XP* restringem o acesso às portas de I/O de um PC no modo usuário. Em 2001, *Thomas Franzon* desenvolveu um aplicativo de fácil utilização denominado *Userport* que possibilita o acesso a essas portas, no modo usuário [91], [92]. Assim, o usuário pode habilitar o endereço I/O que precisa utilizar. No caso da porta paralela, o endereço dos registradores para LPT1 é 0x37A.

O *Userport* vem com os códigos fonte em Visual C++ acessíveis ao usuário e pode ser facilmente obtido na Internet. As etapas necessárias à instalação do *Userport* estão no Apêndice D. Em sistemas operacionais *Linux* e *Windows® 9X*, não é preciso o emprego do *Userport*.

Após o aplicativo ter sido corretamente instalado é suficiente rodar o arquivo executável e configurar o endereço 0x37A da seguinte forma:

a) Na janela esquerda digitar o valor 0x378-0x37A e pressionar o botão "*Add*". Posteriormente, remover os endereços "*default*": 200-37F, 3BC-3BF e 3E8-3FF, usando "*Remove*";

b) Na janela direita digitar o valor 0x378-0x37A e pressionar o botão "*Add*", logo depois, remover os outros endereços.

c) Pressionar o botão "*Start*".

Na Figura 6.20 são mostradas as etapas mencionadas [92].

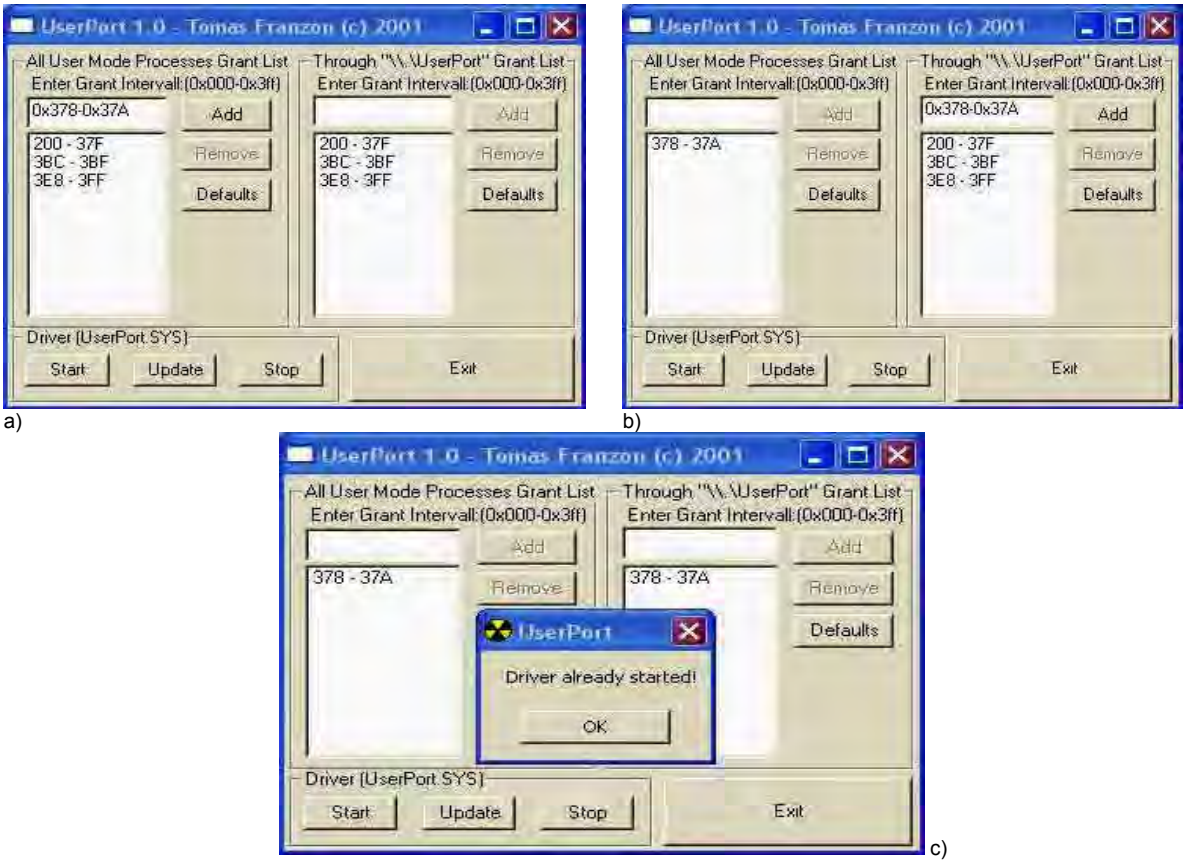


Figura 6.20. Etapas para a configuração do aplicativo *UserPort*.

6.3.7 - API NET

Java NET é uma API para aplicações em ambiente de rede, que possibilita a criação de conexões *sockets* TCP/IP [93]. O conceito de *socket* foi introduzido na seção 2.5.2 do Capítulo 2.

Através da API NET podem ser desenvolvidas aplicações cliente-servidor, tanto para Intranets como para Internet. O modelo de comunicação é ilustrado na Figura 6.21.

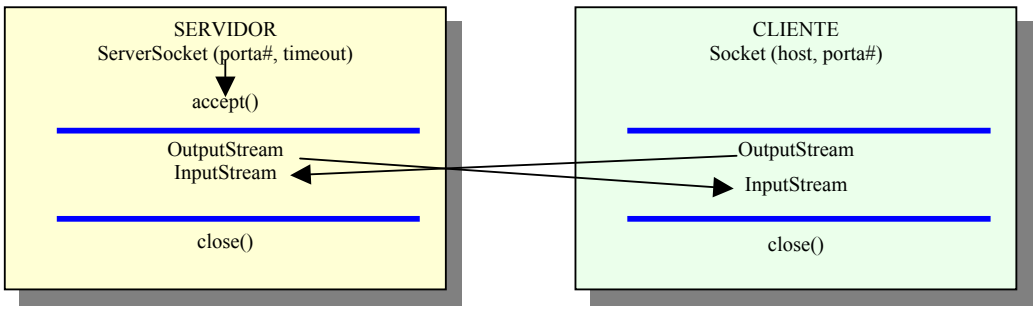


Figura 6.21. Modelo cliente-servidor baseado em *sockets*.

O modelo de comunicação elementar através de *sockets* funciona da seguinte maneira:

a) O servidor estabelece uma conexão em uma porta e aguarda a conexão de um cliente por um determinado período de tempo (*timeout*).

b) Quando um cliente solicita uma conexão, o servidor abre a conexão *socket* usando o método *accept()*. O cliente estabelece a conexão através da porta que lhe foi indicada.

c) O cliente e o servidor interagem por meio de fluxos de entrada e saída pertencentes ao pacote *Java io*.

Com a finalidade de ilustrar o funcionamento básico do Java NET apresenta-se o código fonte de uma aplicação servidor na Figura 6.22, e da aplicação cliente na Figura 6.23.

```
/*Servidor*/
import java.net.*;
import java.io.*;

class Servidor {

 ServerSocket s = (ServerSocket)null;
 Socket s1;
 ObjectOutputStream s1saida;

    public static void main( String args[] ) {

        Servidor conexao = new Servidor();
        conexao.criarSockets(); }

    public void criarSockets() {

        //Estabelecer o servidor no socket 4323 e esperar 10 segundos
        try {
            s = new ServerSocket( 4323,10 );
            System.out.println("Socket criado");
        } catch( IOException e ) {
          System.out.println("O socket nao foi criado" + e);
        }

     //Executar um "loop" de escutar/aceitar
        while( true ) {
            try { //tentativa
                //Esperar para aceitar uma conexão
                s1 = s.accept();
                System.out.println("Um cliente conectado");

                //Obter um fluxo de saída associado com o socket
                s1saida = new ObjectOutputStream( s1.getOutputStream() );

                //Enviar o texto para o cliente
                s1saida.writeObject ( "testando a conexao" );
                s1saida.flush();
                //Fechar a conexão
                s1.close();
            } catch( IOException e ) {
              System.out.println("O Texto nao foi enviado" + e); }
                }
        }
    }
```

Figura 6.22. Código fonte de um servidor básico baseado em *sockets*.

```
/*Cliente*/

import java.net.*;
import java.io.*;

class Cliente {

static Socket s;
static ObjectInputStream  s_in;
static String mensagem;

public static void main( String args[] ) throws IOException {


        //Abrir uma conexão na porta 4323 em localhost
        s = new Socket( "localHost", 4323 );

        //Configurar o fluxo de entrada para objetos
        s_in = new ObjectInputStream( s.getInputStream() );

        //Ler mensagem enviada pelo cliente
        try {
        mensagem = (String) s_in.readObject();
        //Imprimir a mensagem na tela
        System.out.print( mensagem );
        } catch (Exception e) {};

        //Ao acabar, fechar a conexão
        s.close();
        }
   }
```

Figura 6.23. Código fonte de um cliente básico baseado em *sockets*.

Na Figura 6.24 é mostrada, de forma seqüencial, a execução do servidor, informando que um *socket* foi criado, aceitando uma conexão cliente e a tela do lado do cliente mostrando a mensagem “testando a conexão”, enviada pelo servidor.

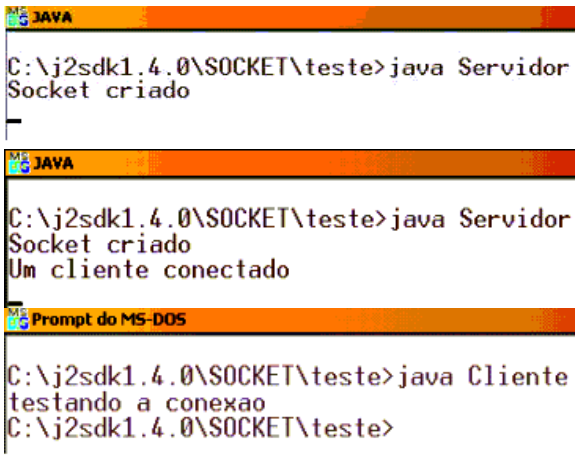


Figura 6.24. Execução do servidor e do cliente.

Como analisado na seção 6.3.4, com o pacote RMI é possível implementar aplicações cliente-servidor em ambientes distribuídos, entretanto, na chamada de métodos remotos é necessária a implementação de uma interface remota. Este fato faz com que aplicações RMI sejam mais lentas do que as aplicações baseadas em *sockets*. Aplicações desenvolvidas através de Java NET são mais eficientes porque a interação entre cliente e servidor é direta [93].

6.3.8 – Desempenho da tecnologia Java

A tecnologia Java apresenta um conjunto de características interessantes para o desenvolvimento de sistemas, em diferentes áreas da engenharia. Dentre essas características têm-se: a orientação a objetos, a robustez, a portabilidade e um amplo conjunto de bibliotecas de classes. Em contrapartida, apresenta alguns pontos fracos para sua utilização em tais sistemas, como: não determinismo, tamanho de código e acesso ao *hardware*.

A diferença de outras linguagens, Java é interpretada, introduzindo sua característica de multiplataforma. A portabilidade não vem sem um preço, pois a interpretação dos *byte-codes* gera retardos na execução dos programas, que dependendo da aplicação, podem ser inaceitáveis. Entretanto, este fato não deve ser interpretado como um assunto geral e impossível de ser solucionado. Estudos recentes demonstram que, com o advento de novos compiladores, Java pode ser tão rápida quanto C++ em questões relacionadas com execução de programas, alocação de memória e acesso aos recursos do sistema [94]. Com o desenvolvimento constante da tecnologia Java, muito provavelmente, não demorará muito tempo para que suas deficiências venham a ser superadas.

Para desenvolvimentos na área de automação e controle, já existem grupos direcionando o problema de aplicações em tempo real [95], [96].

Cumpre salientar também que a *Sun Microsystems* fornece a plataforma J2ME (*Java 2 Platform Micro Edition*™) que tem como objetivo atender as necessidades vinculadas com o desenvolvimento de eletrodomésticos e dispositivos embarcados [88].

6.4. Comentários Finais sobre o Capítulo 6

Foram apresentados, de forma sucinta, os aspectos relevantes das tecnologias empregadas no presente projeto para o desenvolvimento e implementação de um nó IEEE 1451 e sua conexão

em ambiente de rede. Além das características gerais de cada tecnologia e suas vantagens, comentou-se brevemente sobre aspectos de desempenho, com a finalidade de que o leitor tenha presente, nos próximos Capítulos, quais são as principais limitações das tecnologias utilizadas e como elas podem influenciar no projeto.

Dado o caráter abrangente da pesquisa envolvida neste trabalho para a implementação do nó IEEE 1451, cumpre salientar que o objetivo do trabalho não é a avaliação de desempenho do sistema desenvolvido, mas a detecção de possíveis problemas e limitações para que possam ser analisadas em futuros trabalhos do grupo de pesquisa.

No próximo Capítulo, sobre materiais e métodos, será explicado como as tecnologias apresentadas no Capítulo 6 foram utilizadas na implementação do nó IEEE 1451.

Capítulo 7

Projeto do Nó IEEE 1451

Neste capítulo são apresentados e descritos, os materiais e os métodos utilizados durante o desenvolvimento do trabalho, abordando-se o desenvolvimento do STIM, o desenvolvimento do NCAP e a conexão do conjunto STIM-NCAP em um ambiente de rede funcionando com base no padrão IEEE 802.3 e protocolo TCP/IP.

A proposta de implementação é apresentada através da Figura 7.1, onde é possível observar o NCAP baseado em um PC, utilizado em conjunto com um dispositivo gerenciador de protocolo implementado com uma FPGA, se comunicando com o PC através da porta paralela. O STIM da Figura 7.1, conectado ao NCAP por meio da TII, é um modulo implementado com uma FPGA, contendo dois canais transdutores: um sensor de temperatura de uso geral e um ventilador agindo como atuador, ambos com seus respectivos circuitos de condicionamento de sinal. O conjunto NCAP-STIM é conectado a uma rede *Ethernet* 10 Base-T, sendo que, através da rede local ou via Internet, diferentes NCAPs clientes podem acessar a aplicação, para realizar monitoramento e atuação remota das variáveis envolvidas no processo.

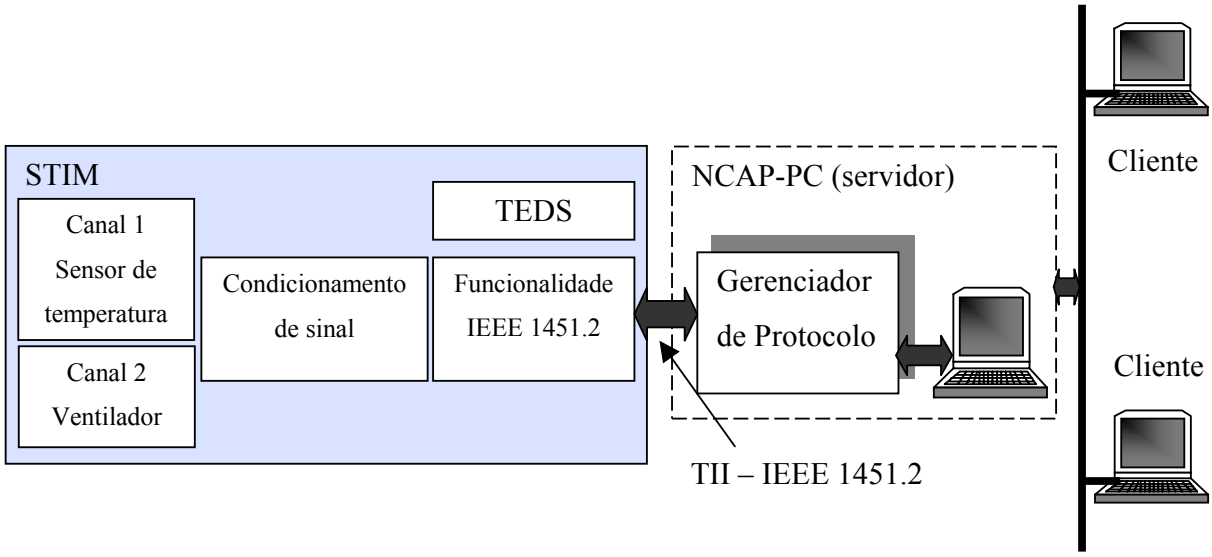


Figura 7.1. Proposta de implementação.

7.1- Desenvolvimento do STIM

O STIM foi desenvolvido utilizando a tecnologia de lógica programável. Empregou-se a linguagem VHDL para descrever o protocolo e os blocos internos que implementam as funcionalidades IEEE 1451.2 e as TEDS. Foram utilizados os ambientes de desenvolvimento *Max+Plus®II e Quartus®II*, ambos da Altera. Com as ferramentas fornecidas pelos ambientes, sintetizou-se o projeto do STIM, que, finalmente, foi programado e implementado com dispositivos PLDs versáteis e de uso geral. Foram necessários circuitos de condicionamento e conversão de sinal, externos ao projeto desenvolvido em VHDL, no entanto, esses módulos também fazem parte do STIM.

7.1.1 - Arquitetura Implementada

A maioria das aplicações em conformidade com o padrão IEEE 1451.2 têm sido implementadas com ASICs (*Application Specific Integrated Circuits*), microcontroladores e placas comerciais específicas [22], [23], [25], [63], [97]. Em todos os casos, as estruturas TEDS foram alocadas em uma memória externa ao processador considerado.

A utilização da tecnologia de lógica programável na implementação de um STIM é uma alternativa pouco explorada, em particular, o emprego de dispositivos de lógica programável versáteis e de uso geral. Girerd et al., em [26], implementaram um nó IEEE 1451, com um STIM baseado em FPGA de arquitetura *multicore*, de 200.000 portas típicas e um NCAP baseado em uma solução comercial. Para aplicações envolvendo um pequeno grupo de canais transdutores, a arquitetura *multicore* pode ser uma alternativa antieconômica, além de complicada no que diz respeito à sua utilização. Castro et al., em [29], implementaram um STIM de dois canais transdutores com um dispositivo FPGA de mais de 100.000 portas típicas.

Em contraste com as aplicações citadas, neste trabalho foi implementado um STIM de dois canais transdutores, sendo que o mesmo projeto foi testado com três tipos de FPGAs de uso geral, de 20.000, 50.000 e 100.000 portas típicas a fim de analisar os recursos necessários para sua implementação com PLDs de baixo custo.

Na Figura 7.2a, é mostrada a arquitetura simplificada do STIM implementado e, na Figura 7.2b, mostram-se os módulos de *software* necessários. Todos os blocos funcionais internos do

STIM, assim como os protocolos associados à TII foram descritos em VHDL empregando–se descrição estrutural e comportamental.

O *software* necessário à implementação do STIM foi dividido em diferentes módulos. Dessa maneira foi possível projetar uma arquitetura flexível e testar o funcionamento de cada bloco funcional antes da implementação da hierarquia final.

Foram implementados os seguintes módulos: controle da TII, endereçamento, registradores de estado e requisição de serviço, memória de programa para alocar as estruturas TEDS, interface com os transdutores e uma rotina central que gerencia o fluxo de programa. Os circuitos de condicionamento e conversão de sinal são externos, entretanto, fazem parte do STIM pelo fato das características dos canais transdutores estarem gravadas nas TEDS.

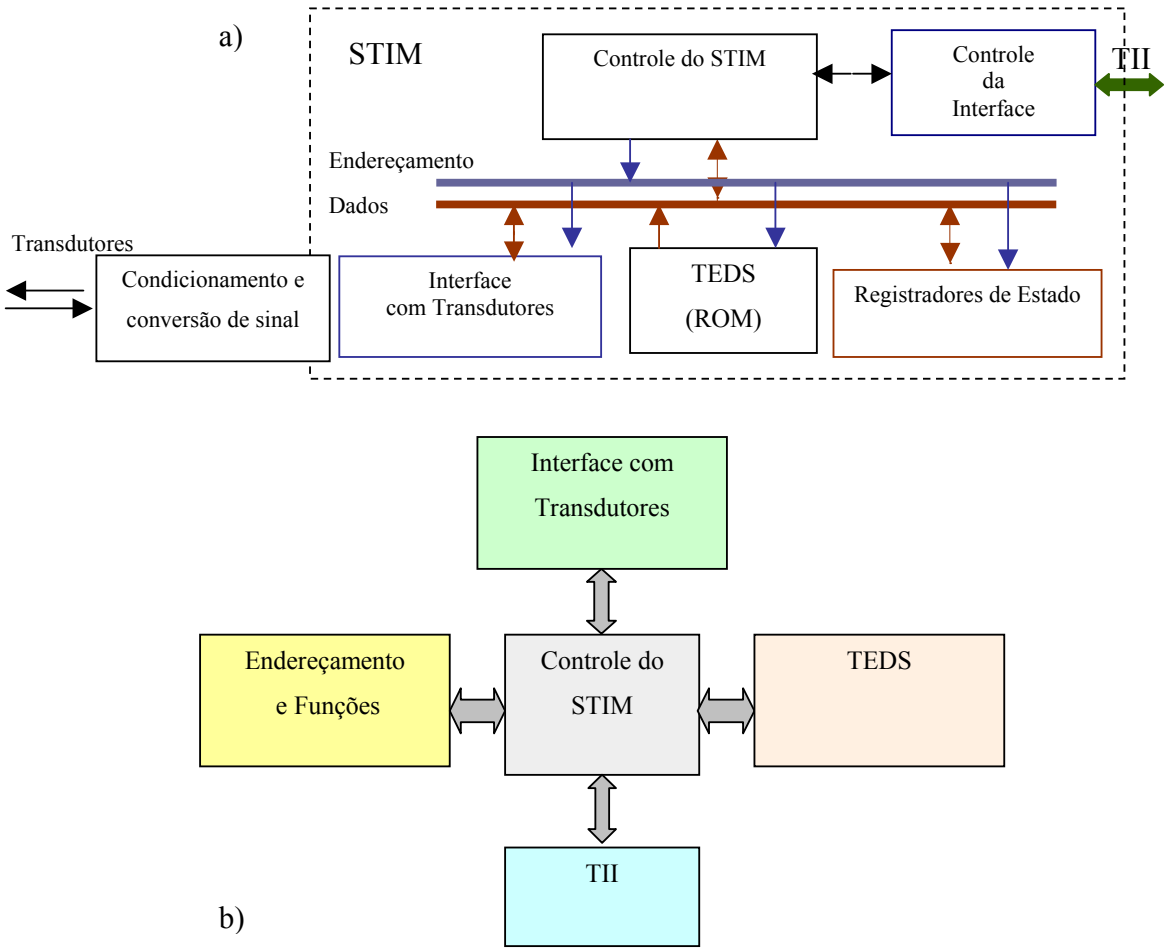


Figura 7.2. a) STIM implementado, arquitetura simplificada; b) Módulos de *software* necessários.

a) Motivação para o Emprego da Tecnologia de Lógica Programável

A escolha da tecnologia de lógica programável para o desenvolvimento do STIM, constitui uma alternativa mais flexível, se comparada com a implementação através de microcontroladores, fundamentalmente porque a linguagem VHDL é padronizada, possui independência tecnológica e possibilita a reutilização de códigos.

Geralmente, no caso dos microcontroladores, emprega-se a linguagem *assembly* na descrição dos programas correspondentes. Esta linguagem é de baixo nível e, geralmente, varia de fornecedor para fornecedor. Desta maneira, o comportamento de um sistema complexo descrito através de um programa em *assembly*, dificilmente pode ser utilizado com outro microcontrolador sem um elevado grau de esforço para readaptá-lo ao novo ambiente. Contudo, existe a possibilidade dos programas serem escritos em C ou C++. Entretanto, transformar esses códigos, em *assembly*, para um microcontrolador específico, requer de um compilador.

Algumas soluções comerciais para implementação de sistemas baseados no padrão IEEE 1451 podem ser empregadas. Teoricamente, diferentes soluções fornecidas por diferentes fabricantes deveriam interoperar, no entanto, as placas comerciais introduzidas especificamente com essa finalidade podem resultar em uma alternativa antieconômica e podem apresentar algumas características de natureza proprietária [24], [28].

Assim, pensando-se na construção de um sistema com elevado grau de modularidade, aberto e baseado em padrões, fato que é marcado pela utilização do padrão IEEE 1451, a alternativa de implementação de um STIM através da tecnologia de lógica programável, mostra-se como uma alternativa mais adequada e merece ser explorada em detalhe.

7.1.2 - Arquitetura Detalhada

O diagrama da arquitetura detalhada do STIM implementado é apresentado na Figura 7.3. Na seqüência, cada módulo componente do STIM é explicado detalhadamente. No apêndice E é mostrada esta arquitetura, obtida a partir da entrada gráfica do ambiente de síntese.

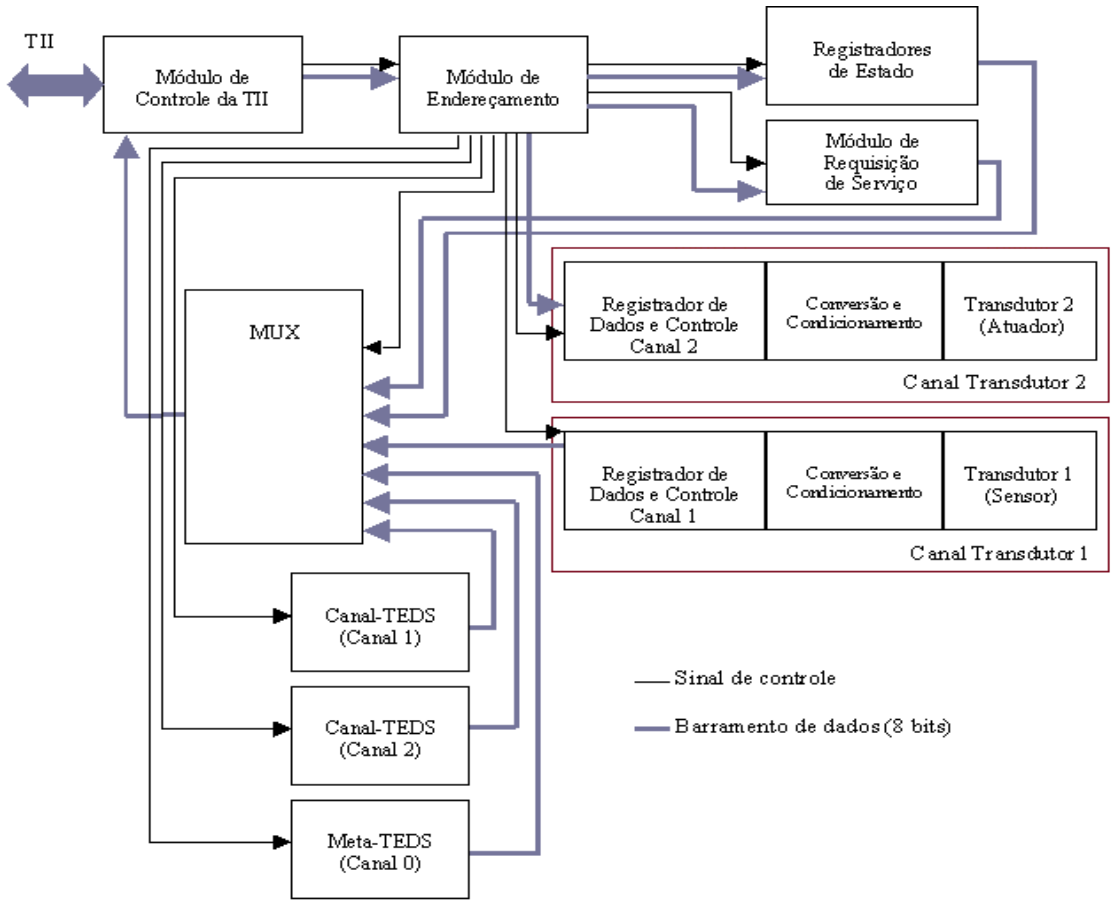


Figura 7.3. STIM implementado, arquitetura detalhada.

7.1.2.1 - Módulo Central

O módulo central da arquitetura proposta consiste em três blocos funcionais diferentes:

a) Módulo de Controle da TII;

b) Módulo de Endereçamento;

c) Multiplexador.

a) Módulo de Controle da TII

O módulo de controle da TII é responsável por controlar o fluxo de dados e sinais de controle através da interface TII, do lado do STIM, implementando o comportamento do fluxograma apresentado na Figura 5.8. O protocolo de comunicação entre o NCAP e o STIM é gerenciado pelo NCAP. O módulo de controle da TII modifica o sinal NACK em resposta a um sinal de

disparo por parte do NCAP, recebe o sinal de habilitação NIOE, agindo em conseqüência, e determina as transições no sinal NACK, toda vez que um pacote de 8 *bits* é transmitido ou recebido.

O módulo de controle obtém os dados que entram via DIN, de forma série, e os dispõe, de forma paralela, para processamento interno no STIM. O módulo também é responsável por obter dados internos do STIM, de forma paralela, e dispô-los para o NCAP via DOUT, de forma série.

Os dados que entram através de DIN são divididos internamente e colocados em quatro barramentos de 8 *bits*, endereço de função[30], endereço de canal e operações de escrita 1 e 2. Esses barramentos, por sua vez, são conectados com o módulo de endereçamento, a fim de que este, atue em conseqüência.

Por seu turno, um multiplexador disponibiliza para o módulo de controle da TII, tanto os dados internos associados com um canal transdutor, como os dados contidos nas TEDS. Assim, o módulo de controle envia os dados através da TII, via DOUT.

Além dessas funções, o módulo de controle da TII gera vários sinais de controle que são utilizados para gerenciar processos internos do STIM, controlar os módulos TEDS e a aquisição de dados correspondentes aos canais transdutores. Na Figura 7.4 é apresentado o símbolo do módulo de controle, denominado *protoc_ctrl* e gerado pelo ambiente de síntese.

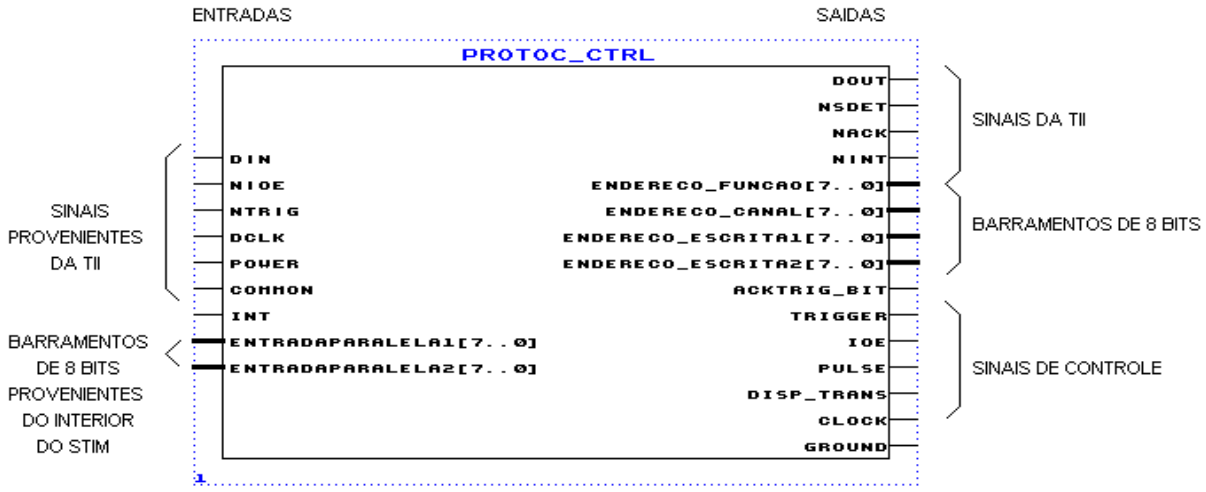


Figura 7.4. Módulo de controle da TII.

[30] De acordo com o padrão utiliza-se a denominação endereço de função ou endereço funcional. O endereço é composto por um código de 8 *bits*. Em virtude de que este valor sempre é endereçado desde o NCAP para o STIM, o termo endereço de função resulta mais conveniente do que código de função.

b) Módulo de Endereçamento

Este módulo é responsável por realizar o endereçamento lógico de acordo com as funcionalidades do padrão IEEE 1451.2. O módulo recebe de *protoc_ctrl*, os endereços de função, de canal e de escrita, se houver. O processamento é feito em função dos valores desses endereços. Por exemplo, se o endereço de função for 160 e o canal o número 1, o módulo de endereçamento disponibiliza, para o módulo multiplexador, os valores contidos no bloco Canal-TEDS1.

Na Figura 7.5a, é mostrado o bloco denominado *enderec*, que gera também vários sinais de controle para atualizar registradores e para controlar as TEDS e a interface com os transdutores.

c) Multiplexador

O bloco multiplexador, apresentado na Figura 7.5b, possui um conjunto de entradas de controle, um conjunto de barramentos de entrada de 8 *bits* e dois barramentos de saída, também de 8 *bits*. Dependendo de qual das entradas estiver em nível '1', o conteúdo do barramento de entrada correspondente passa para a saída do multiplexador, prestes a ser enviado ao bloco *protoc_ctrl*.

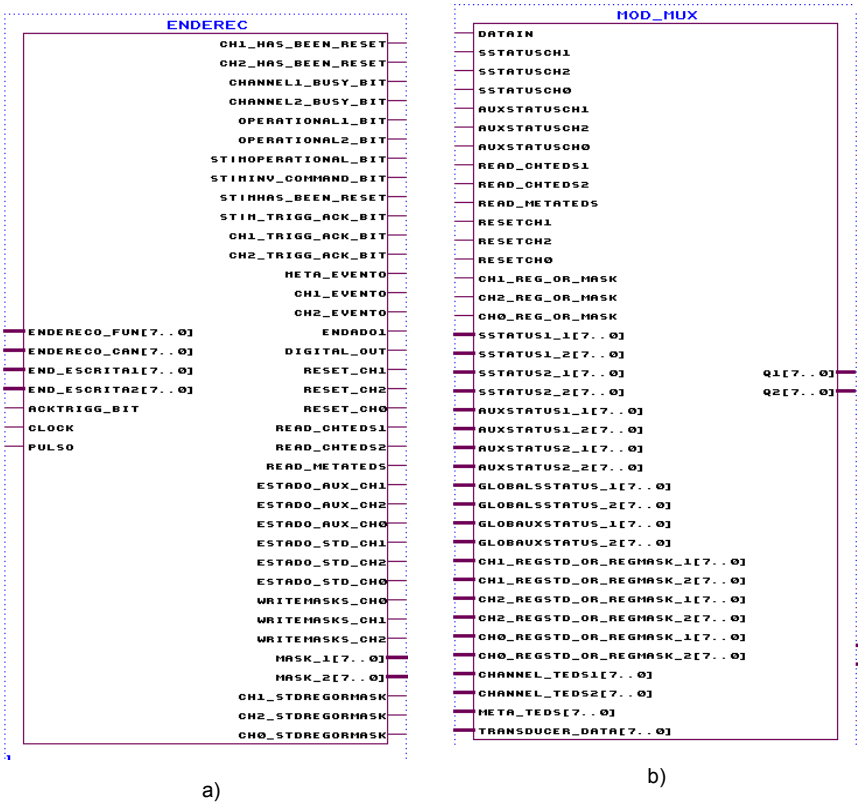


Figura 7.5. a) Módulo de Endereçamento; b) Módulo multiplexador.

Cumpre salientar que, associada à lógica do módulo central está o protocolo IEEE 1451.2, que foi descrito integralmente em VHDL. O protocolo inicia-se com uma comunicação do tipo *handshake* entre o NCAP e o STIM para sincronizar os módulos e, logo depois, uma comunicação do tipo *half-duplex* a fim de transferir os dados associados com os canais transdutores. Os códigos fonte são fornecidos no disquete anexado ao final do trabalho.

7.1.2.2 - Estruturas TEDS

Cada estrutura TEDS implementada foi alocada em uma memória do tipo ROM parametrizável, que faz parte da biblioteca de módulos parametrizáveis (LPM – *Library of Parameterized Modules*) pertencente ao ambiente de síntese.

Uma função parametrizável utiliza parâmetros reconfiguráveis a fim de atingir escalabilidade e adaptabilidade [98], [99]. Este fato pode ser analisado no código fonte da Figura 7.6, pertencente à descrição do bloco Canal-TEDS1. No código apresentado é utilizada a descrição estrutural. Note-se que o parâmetro *lpm_file* estabelece uma ligação com o arquivo *channelteds1.mif*, que é um arquivo de iniciação de memória onde está codificado, através de uma entrada textual, o conteúdo completo do bloco TEDS considerado.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
LIBRARY lpm;
USE lpm.lpm_components.ALL;
ENTITY channel_teds1 IS
PORT(
clk     : IN  std_logic;
address : IN  std_logic_vector(6 DOWNTO 0);
q       : OUT std_logic_vector(7 DOWNTO 0));
END channel_teds1;
ARCHITECTURE comp OF channel_teds1 IS
BEGIN
--Utilização de memória rom parametrizável
instanciar: lpm_rom
GENERIC MAP(
--Definição dos parâmetros da memória
lpm_widthad => 7,
lpm_width   => 8,
lpm_file    => "channelteds1.mif",
lpm_address_control => "UNREGISTERED")
--Mapeamento de entradas e saídas
PORT MAP
(address => address, outclock => clk, q => q);
END comp;
```

Figura 7.6. Descrição VHDL da memória com capacidade de suportar as TEDS.

Foram implementadas as estruturas Meta-TEDS e Canal-TEDS, para os canais 1 e 2. O comportamento necessário à movimentação dos dados contidos nas TEDS foi descrito em VHDL. O endereçamento das posições de memória foi realizado da maneira mais prática e simples possível, através de um contador que dispara o processo toda vez que acontece uma transição no sinal NACK. Deste modo é possível enviar múltiplos pacotes de 8 *bits* através do sinal DOUT da interface, quando for requisitada a leitura das TEDS. Na figura 7.7 é apresentada a configuração utilizada.

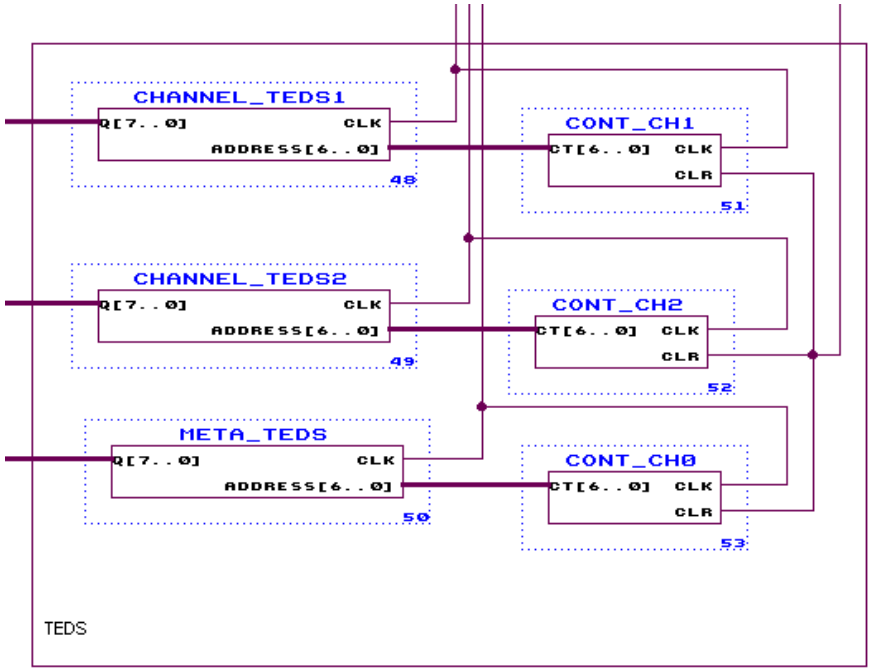


Figura 7.7. Módulo que implementa as estruturas TEDS.

Cada bloco TEDS precisa em torno de 100 *bytes*, portanto, é suficiente ter endereços de entrada de 7 *bits*, pois $2^7 = 128$, ou seja, 128 posições de memória diferentes. O barramento de saída da memória é de 8 *bits*, assim, a capacidade de memória é de 1024 *bits*, pois 128 x 8 = 1024. Considerando-se então, 128 *bytes* por cada canal implementado, para dois canais somados ao canal zero, serão necessários 3072 *bits* de memória para alocar as TEDS. A análise do impacto destes valores e do aumento de canais transdutores sobre os recursos necessários será feita na seção 8.1.1 do próximo capítulo, para os dispositivos lógicos programáveis considerados neste trabalho.

Toda vez que acontece uma transição no sinal NACK, o módulo *protoc_ctrl* envia um sinal de controle para os blocos TEDS, indicado como *clk*, na Figura 7.7.

De modo ilustrativo, na Figura 7.8, mostra-se um exemplo de simulação. Na primeira transição do relógio dispara-se o contador com o valor "0000001_2" ou "1_{10}" que corresponde à primeira palavra de 8 *bits* escrita em memória, neste caso "00000000". Este valor fica disponível para o módulo multiplexador, que fornecerá os valores para o módulo *protoc_ctr*l quando os endereços processados pelo módulo de endereçamento forem 160 e canal 0 (leitura do Meta-TEDS), ou 160 e canal 1 (leitura do Canal-TEDS 1), ou 160 e canal 2 (leitura do Canal-TEDS 2). Já na quarta transição do sinal NACK, o sinal de controle dispara o contador com o valor "0000100_2" ou "4_{10}" que corresponde à quarta palavra de 8 *bits* escrita em memória: "01011100_2" ou "92_{10}". O processo se repete até transmitir, de forma completa, a estrutura TEDS correspondente, através do sinal DOUT da TII, via intermediação do módulo *protoc_ctrl*.

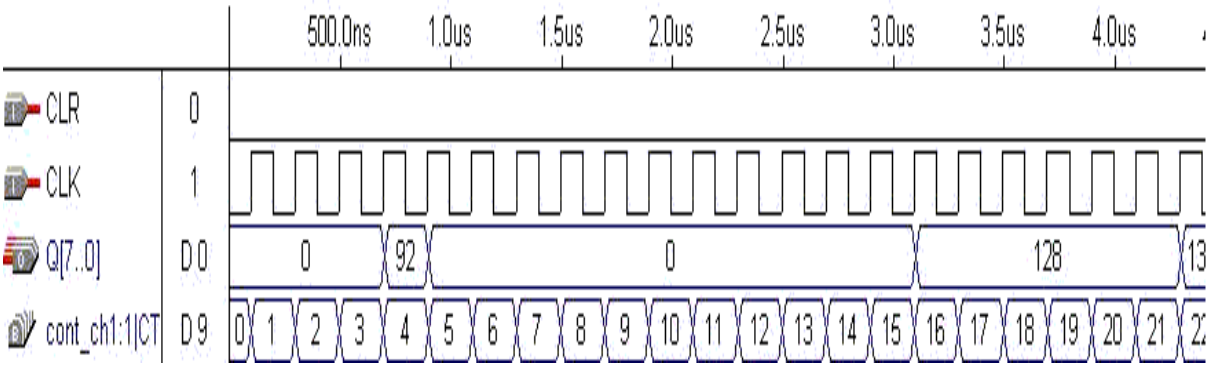


Figura 7.8. Tratamento das TEDS; exemplo de simulação.

7.1.2.3 - Registradores de Estado e Módulo de Requisição de Serviço

De acordo com o padrão IEEE 1451.2, o STIM deve conter um registrador de estado padrão e um registrador auxiliar de estados para o canal 0, além disso, associado com cada canal transdutor implementado, deve existir um registrador de estado padrão e outro auxiliar. Na Figura 7.9, é mostrado o conjunto de registradores associados com o canal 1.

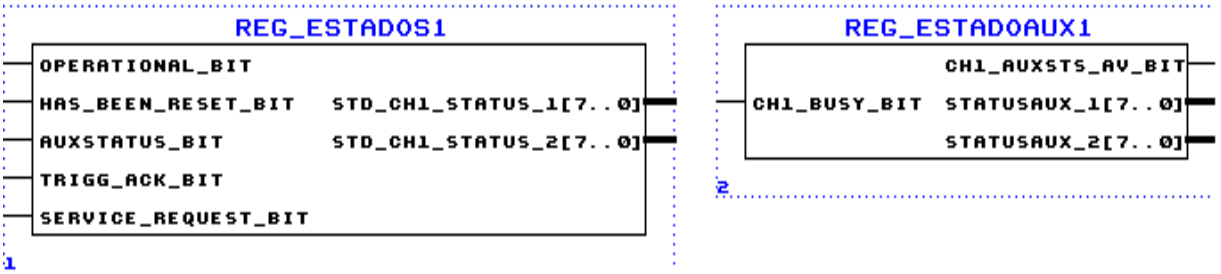


Figura 7.9. Registradores de estados, padrão e auxiliar.

De acordo com o processamento interno no STIM, os *bits* dos registradores podem sofrer modificações no seu estado. Através do endereço de função 130 aplicado ao canal desejado, o NCAP tem acesso a esses registradores. Por exemplo, o *bit* correspondente a *Channel has been reset* no estado lógico '1' indica que o canal foi "resetado".

Por seu turno, apenas um *bit* auxiliar foi considerado, que indica canal ocupado toda vez que for processado um comando. Neste caso *channel busy bit* assume o estado lógico '1' no registrador auxiliar de estados.

No caso de serem implementados vários canais, deve-se realizar a operação OR lógica entre todos os *bits* indicadores de canal ocupado. Caso esse resultado seja 1, então, o bit *OR of channel busy bits* do registrador global auxiliar de estados assumirá o valor '1'.

Quanto ao módulo de requisição de serviço, a sua função é a de gerar uma interrupção, que é enviada para o NCAP através do sinal NINT da TII. O módulo é responsável por realizar operações lógicas entre os registradores de estado e os registradores de máscaras de interrupção. As máscaras com todos os *bits* em '0' não permitem gerar interrupções. Em contraste, as máscaras com todos os *bits* em '1', possibilitam a geração de interrupções. As máscaras podem ser escritas através do endereço de função 5 aplicado ao canal correspondente, os valores de escrita são enviados através do endereço de escrita 1 e do endereço de escrita 2.

O módulo implementado pode ser observado no diagrama detalhado do STIM, no apêndice E.

7.1.2.4 - Interface com os Transdutores

A interface com os transdutores opera com base em registradores de dados e sinais de controle provenientes do bloco *protoc_ctrl* e do bloco *enderec*. O bloco *mod_adq*, mostrado na Figura 7.10 é utilizado para colocar os dados fornecidos pelo conversor A/D em um registrador de

dados. Se um evento de disparo acontecer, o módulo *protoc_ctrl* gera um sinal (*disptrans*) que habilita o registrador. Os dados estarão disponíveis para serem enviados ao NCAP, apenas quando o endereço de função for 128 para o canal 0 ou 1. Isto faz com que, toda vez que aconteça um evento de disparo, o registrador possa atualizar os valores, mas apenas quando o endereço de função for igual a 128, os dados poderão ser processados.

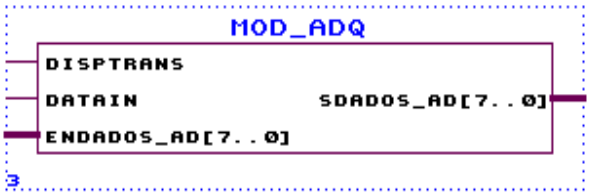


Figura 7.10. Módulo de interface com transdutores.

No canal transdutor 1, usou-se um conversor A/D[31] de 8 *bits* modelo AD0804 de aproximações sucessivas, conectado a um sensor de temperatura integrado, de uso geral, modelo LM35 com resolução de 10 mV/ºC. Entre o sensor e o conversor A/D foi necessário implementar um circuito de condicionamento a fim de adequar o sinal fornecido pelo sensor[32].

No canal transdutor 2, usou-se um ventilador convencional para uso interno em gabinetes de microcomputadores, agindo como atuador.

O módulo *enderec* gera um sinal digital padrão TTL (*Transistor-Transistor Logic*) que é utilizado para acionar o ventilador via intermediação de um circuito de condicionamento, pois o atuador opera com 12 Vdc.

Embora o canal 1 seja composto por um conversor A/D de 8 *bits*, pode ser utilizado com 12 *bits* ou mais, pois os módulos descritos em VHDL possuem um parâmetro genérico onde é possível definir a quantidade de *bits* dos barramentos. Salienta-se que os barramentos são parametrizáveis mas não reconfiguráveis. A fim de modificar os valores é necessário modificar os parâmetros dos barramentos de dados manualmente, entretanto, isto não requer grandes esforços em virtude da flexibilidade da descrição VHDL. Os endereços de função, canal e escrita, e os barramentos associados com os registradores de estado e blocos TEDS continuam sendo de 8 *bits*, pois estes valores dependem do padrão IEEE 1451.2 e não do projetista.

[31] As características do conversor A/D utilizado encontram-se no Apêndice F.
[32] Detalhes do circuito de condicionamento encontram-se no Apêndice F.

7.1.3 – Suporte de *Hardware* Empregado

Com o intuito de analisar a possibilidade de serem empregados PLDs versáteis e de uso geral, o STIM foi implementado e testado com três tipos diferentes de FPGAs, cujas características principais são apresentadas na Figura 7.11 [100], [101], [102].

Todo STIM precisa de 10 sinais de entrada/saída, para que possa ser conectado ao NCAP, através da TII. Além disso, o STIM proposto neste trabalho possui um canal transdutor com uma entrada digital de 8 *bits* e um canal transdutor composto por uma saída digital. Ao todo são necessários 19 pinos de entrada/saída.

Cada estrutura TEDS necessita em torno de 100 *bytes* para ser alocada em memória. Como analisado na seção 7.1.2.2, no STIM desenvolvido neste trabalho foram considerados 128 *bytes* por cada canal, totalizando 3072 *bits* para os dois canais transdutores mais o canal zero. Conseqüentemente, do ponto de vista dos recursos de memória disponível e das entradas e saídas necessárias, qualquer uma das alternativas apresentadas na Figura 7.11 é viável. Entretanto, não é possível se conhecer com antecedência, o número de elementos lógicos necessários. Portanto, para o caso proposto, o ponto crítico que determina a escolha entre as alternativas apresentadas é o número de elementos lógicos.

Evidentemente, o aumento do número de canais implementados produz um incremento no número de células lógicas utilizadas. Isto porque, por cada canal acrescentado, incrementa-se o número de registradores e de barramentos internos. Porém, é necessário prever que se fossem implementados, por exemplo, mais três canais contendo sensores com interfaces via conversores de 8 *bits*, seriam necessárias mais 24 entradas/saídas e mais 300 *bytes* de memória.

De modo ilustrativo, para uma aplicação elementar levando em conta dois canais transdutores mais o canal zero são necessários em torno de 830 elementos lógicos com incremento de aproximadamente 150 elementos lógicos por cada canal acrescentado. Um estudo detalhado sobre este item, e sobre a influência do aumento do número de canais transdutores nos FPGAs apresentados, encontra-se na seção 8.1.1 do próximo capítulo.

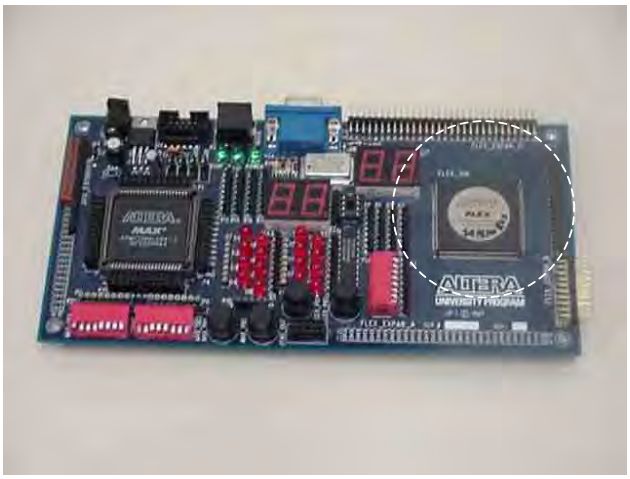
	a) FLEX EPF10K20RC240-4, contido na placa do programa universitário da Altera (UP1). Tecnologia SRAM Quantidade de portas típicas: 20.000 Quantidade de elementos lógicos: 1.152 Pinos de entrada/saída: 189 (níveis de tensão TTL) Capacidade RAM máxima em *bits*: 12.288
	b) ACEX EP1K50TC144-3 [33] Tecnologia SRAM Quantidade de portas típicas: 50.000 Quantidade de elementos lógicos: 2.880 Pinos de entrada/saída: 102 (níveis de tensão TTL) Capacidade RAM máxima em *bits*: 40.960
	c) ACEX EP1K100QC208-3 Tecnologia SRAM Quantidade de portas típicas: 100.000 Quantidade de elementos lógicos: 4.992 Pinos de entrada/saída: 147 (níveis de tensão TTL) Capacidade RAM máxima em *bits*: 49.152

Figura 7.11. Principais características dos PLDs utilizados para implementar o STIM.

[33] As placas apresentadas nos itens b e c foram fornecidas pela PI Componentes.

7.2 - Desenvolvimento do NCAP

O NCAP desenvolvido neste trabalho é composto pelo *hardware* necessário à conexão do módulo com a rede, um dispositivo gerenciador de protocolo, que constitui a interface de *hardware* entre os blocos de *software* do NCAP e a TII e, finalmente, o *software* do NCAP. A arquitetura simplificada do NCAP implementado é apresentada na Figura 7.12.

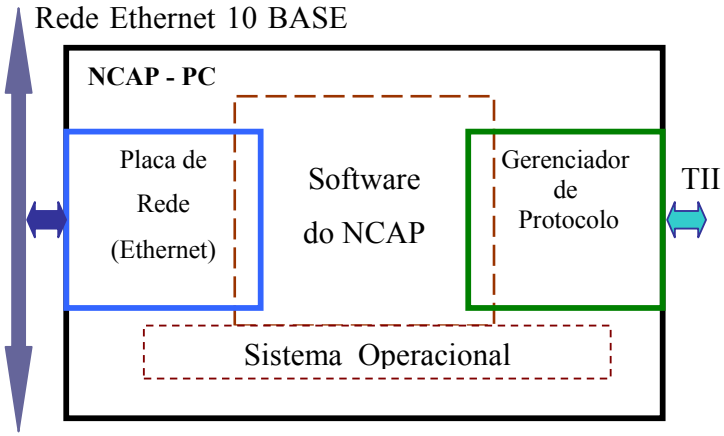


Figura 7.12. Arquitetura do NCAP implementado.

Lee e Schneeman, em [20], utilizaram os recursos de um PC para implementar um NCAP. Porém, modificaram a porta paralela do PC a fim de suportar a interface TII. Posteriormente, não têm aparecido implementações deste tipo. Aplicações com base no padrão IEEE 1451.1 têm sido implementadas com placas comerciais e microcontroladores [23], [25], [26], [27], [30].

O *software* do NCAP é um ponto essencial no desenvolvimento do nó de rede. Neste sentido, os desenvolvimentos científicos pioneiros nesta área têm sido realizados com base na linguagem C e C++ [20], [14]. Já as aplicações feitas com placas comerciais empregaram *softwares* proprietários.

No presente trabalho propõe-se a utilização de um PC convencional junto a um dispositivo Gerenciador de Protocolo implementado com um PLD, para estabelecer a interface entre a porta paralela e a TII. Desta maneira, o gerenciador pode ser conectado diretamente à porta paralela, sem realizar modificações da mesma, possibilitando a utilização de qualquer PC.

A fim de desenvolver o *software* do NCAP propõe-se a linguagem Java, pois é puramente orientada a objeto e seu emprego no desenvolvimento de módulos IEEE 1451 é atualmente pouco explorado. Optou-se por utilizar um PC e a tecnologia Java pelos seguintes motivos:

a) Não é necessário o emprego de uma máquina sofisticada. É necessário apenas que contenha uma Máquina Virtual Java.

b) É possível a implementação de um sistema aberto;

c) O sistema pode ser utilizado como ferramenta didática, pois através do PC é muito mais fácil interpretar o funcionamento do *software* do NCAP;

d) Torna-se mais simples a conexão em rede.

d) Java possui relação direta com as tecnologias associadas à Internet.

e) O Gerenciador de Protocolo possibilita a utilização de qualquer PC e, ao mesmo tempo, trocar o STIM conectado no sistema, por outro, sem comprometer o funcionamento dos módulos.

7.2.1 - Suporte *Hardware* Utilizado

No diagrama da Figura 7.13, são mostrados os elementos de *hardware* utilizados para implementar o nó IEEE 1451. O Gerenciador de Protocolo foi implementado com uma FPGA FLEX 10K20RC240-4. O Gerenciador é conectado entre a TII e a porta paralela de um PC.

Na Figura 7.14 são mostrados: o conector padronizado DB 25 e o mapeamento dos pinos nos registradores internos da porta paralela.

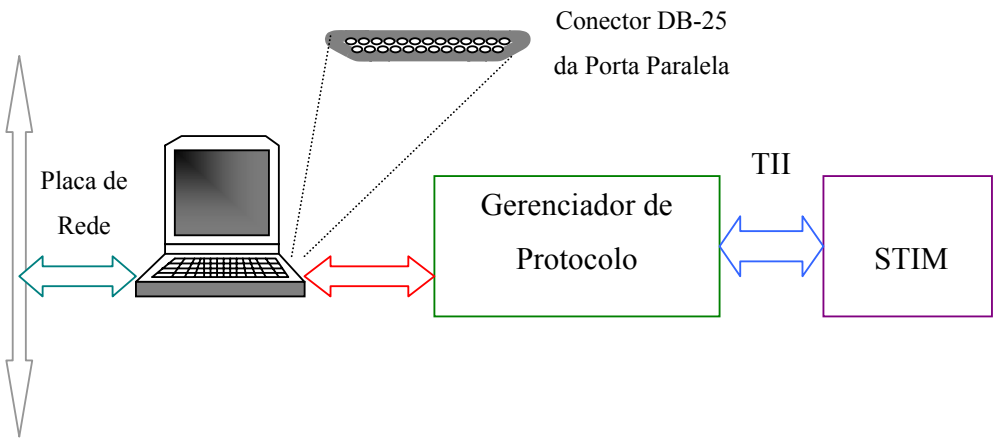


Figura 7.13. Suporte *hardware* do NCAP.

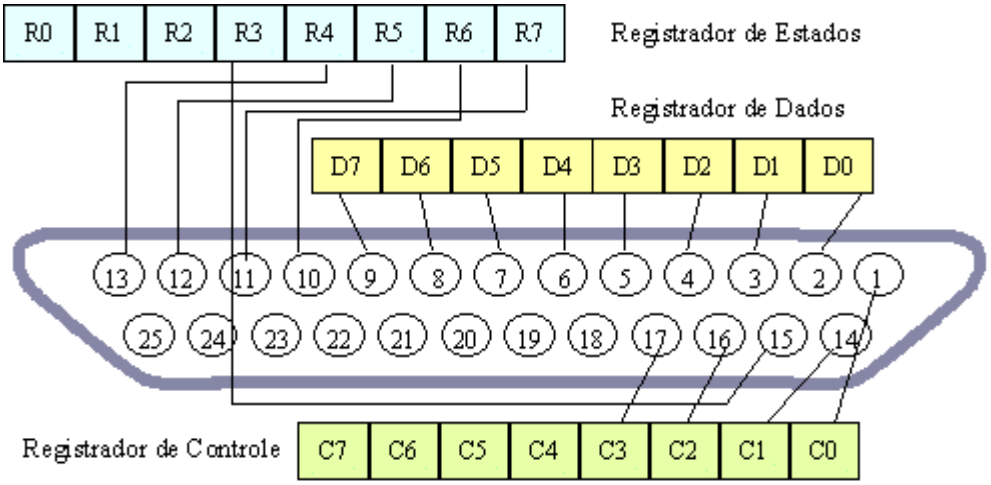


Figura 7.14. Conector DB-25 e registradores internos da porta paralela.

Além da porta paralela tradicional (SPP - *Standard Parallel Port*), os computadores atuais suportam os modos: incrementado (EPP – *Enhanced Parallel Port*) e de capacidade estendida (ECP – *Extended Capability Port*), padronizados através da diretriz IEEE 1284 [103]. Desta maneira é possível a comunicação entre um PC e dispositivos externos a uma velocidade que varia de 1 Mbps até 2 Mbps.

Os modos de operação da porta paralela podem ser configurados por meio do Sistema Básico de Entrada e Saída (BIOS – *Basic Input Output System*) do computador considerado, sendo que neste projeto, o *software* do NCAP utiliza a configuração EPP+ECP.

A fim de conectar o computador à rede é necessária uma placa de interface de rede para suportar a tecnologia *Ethernet* 10 Base T.

7.2.2 - Gerenciador de Protocolo

Gerenciador de Protocolo é a denominação dada, neste trabalho, ao dispositivo que implementa a interface entre o *software* do NCAP e a TII. O comportamento do Gerenciador de Protocolo foi descrito integralmente em VHDL e implementado com um dispositivo PLD.

O gerenciador é responsável por obter os dados e os sinais de controle através da porta paralela, com a finalidade de disparar e gerenciar o processo que estabelece o protocolo IEEE 1451.2, controlando assim, as atividades do STIM. Desta maneira, o Gerenciador de Protocolo recebe os endereços de função, canal e escrita, através da porta paralela do PC e os disponibiliza,

de maneira sincronizada, para a interface TII. Além disso o gerenciador recebe do STIM os dados por meio do sinal DIN e os disponibiliza para tratamento interno no NCAP.

O gerenciador foi desenvolvido com o intuito de fornecer uma solução que possa ser empregada em diversas aplicações e possibilite o modo de operação *plug and play* do STIM, pois o dispositivo pode ser conectado a qualquer PC que implemente a funcionalidade do *software* do NCAP desenvolvido neste trabalho. Introduz-se, deste modo, uma grande flexibilidade no sistema, pois a porta paralela não precisa ser modificada e, além disso, o gerenciador pode ser implementado com um PLD de baixo custo e uso geral.

Foram desenvolvidas e implementadas, duas versões do Gerenciador de Protocolo, a primeira versão possui uma entrada de 8 *bits* de dados e 2 *bits* de controle, e uma saída de 8 *bits* para realizar operações de leitura. A segunda versão possui uma entrada de dados de 8 *bits* e uma saída de 5 *bits* para operações de leitura.

7.2.2.1 - Gerenciador de Protocolo com Entrada e Saída de Oito *Bits*

O Gerenciador de Protocolo com entrada e saída de oito *bits* foi a primeira versão implementada, podendo ser utilizado com sucesso junto com um *software* de gerenciamento da porta paralela baseado na linguagem Phyton[34], que possibilite a leitura e escrita com 8 *bits*.

Esta versão foi desenvolvida com base no conceito de máquina de estados, utilizando a descrição VHDL comportamental. O funcionamento do dispositivo é relativamente complexo e, ao todo, foram necessários 80 estados[35] para atingir o nível de desempenho desejado. Além das entradas e saídas mencionadas, o dispositivo possui um sinal de habilitação e um sinal de relógio, do qual dependem os estados implementados.

O diagrama esquemático desta versão do Gerenciador de Protocolo é mostrado na Figura 7.15. O dispositivo é composto basicamente pelos seguintes blocos: multiplexador e controle de dados, módulo de disparo, detecção do STIM, gerenciador e demultiplexador. Os códigos fonte correspondentes são fornecidos no disquete anexado ao final do trabalho.

---

[34] Phyton: é uma linguagem puramente orientada a objeto e de domínio público, que introduz modularidade e portabilidade. Phyton é interpretada, no entanto, não é necessário compilar os programas, como acontece com Java.

[35] Dada a complexidade do projeto, optou-se por não inserir um diagrama de estados, pois os estados implementados podem ser facilmente analisados e contabilizados a partir dos códigos fonte pertencentes aos diferentes blocos que fazem parte do projeto do gerenciador. Isto é possível graças à flexibildade da descrição em VHDL.

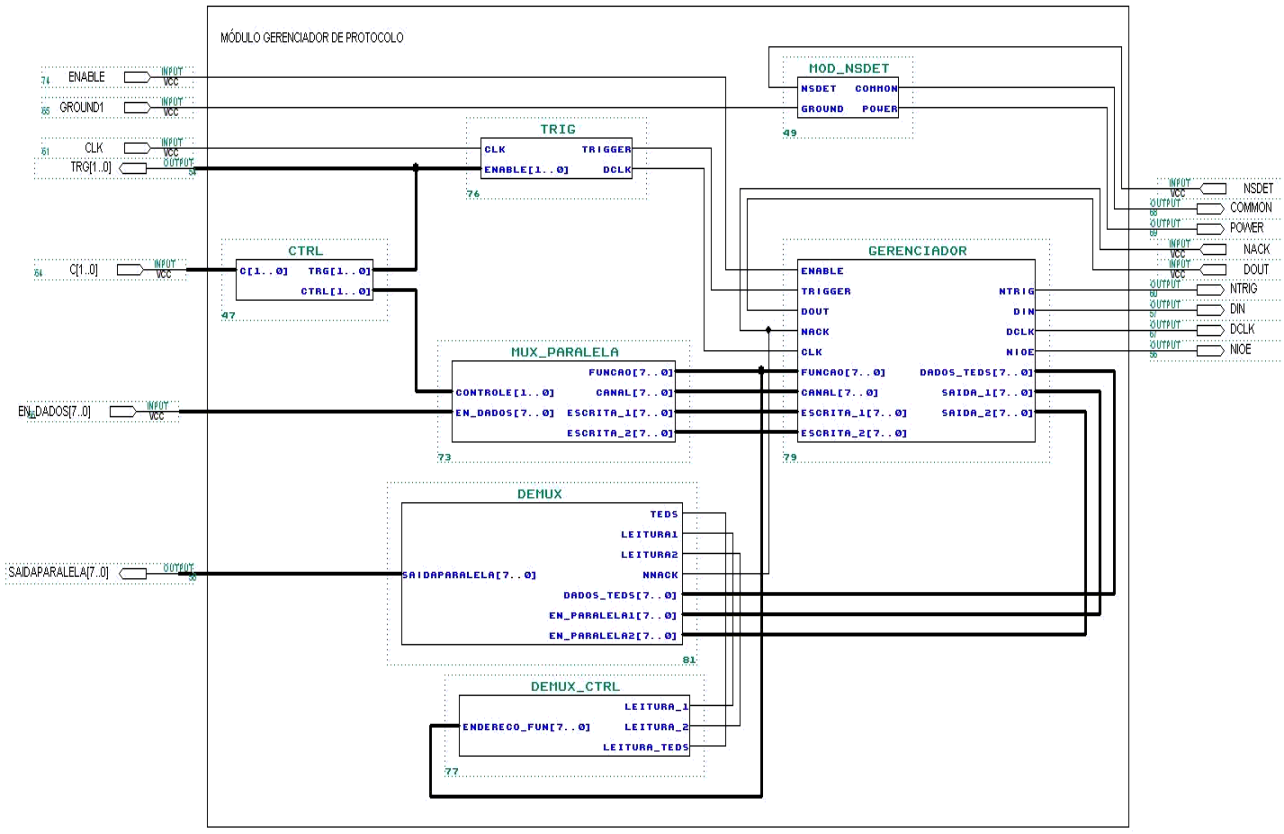


Figura 7.15. Gerenciador de Protocolo com entrada e saída de oito *bits*.

a) Módulo de Multiplexação

Este módulo é composto por um sistema multiplexador preparado para aceitar, da porta paralela, o envio de 10 *bits*, sendo que 8 são de dados (EN_DADOS[7..0]) e 2 são de controle (C[1..0]).

Desta maneira é possível multiplexar os dados de acordo com a configuração mostrada na Figura 7.16. Os blocos funcionais responsáveis por realizar a multiplexação são o *Mux_paralela* e o *Ctrl*, apresentados na Figura 7.17. A interligação entre os módulos pode ser observada na Figura 7.15.

C[1..0] = 01	=> FUNÇÃO [7..0] = EN_DADOS
C[1..0] = 10	=> CANAL [7..0] = EN_DADOS
C[1..0] = 11	=> ESCRITA_1 [7..0] = EN_DADOS
C[1..0] = 00	=> ESCRITA_2 [7..0] = EN_DADOS

Figura 7.16. Multiplexação de dados da porta paralela.

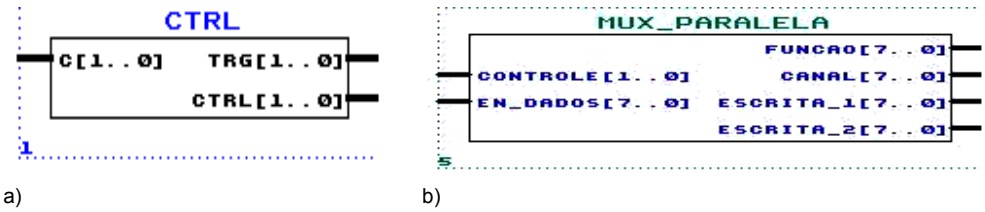


Figura 7.17. a) Bloco de controle, b) Bloco multiplexador.

b) Módulo de Disparo

Quando da porta paralela for enviado um valor de controle igual a '01', o bloco *ctrl* ativa o bloco funcional *trig*, responsável por disparar o *gerenciador*. Assim, o *gerenciador* irá gerar um pulso que sai, através da TII, via NTRIG. O bloco funcional *trig* é mostrado na Figura 7.18.

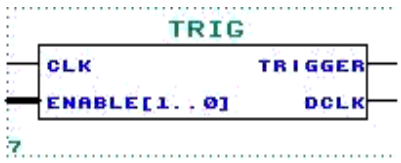


Figura 7.18. Módulo de disparo.

c) Módulo de Detecção do STIM

De acordo com o padrão IEEE 1451.2, o STIM deve ser retirado ou conectado do conector da TII, sem desligar a alimentação do NCAP. O módulo *Mod_Nsdet* fornecerá alimentação (POWER) para o STIM, por meio da TII, apenas quando detectar a sua presença, através do sinal NSDET em nível ativo baixo. Entretanto, o NCAP sempre possui alimentação ativa (PWR), como observado no diagrama da Figura 7.19 a, que ilustra o modo *hot-swap*. O bloco funcional *mod_nsdet* é mostrado na Figura 7.19 b.

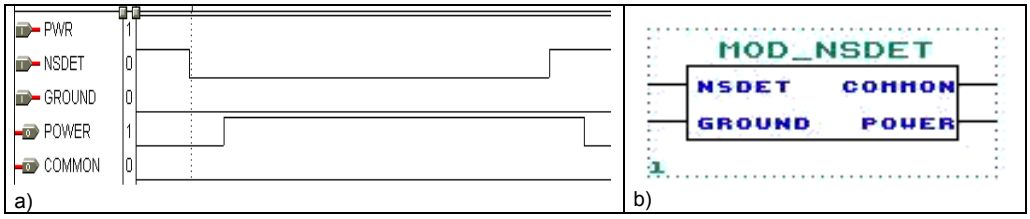


Figura 7.19. a) Modo *hot-swap*, b) Módulo de detecção do STIM.

d) Gerenciador

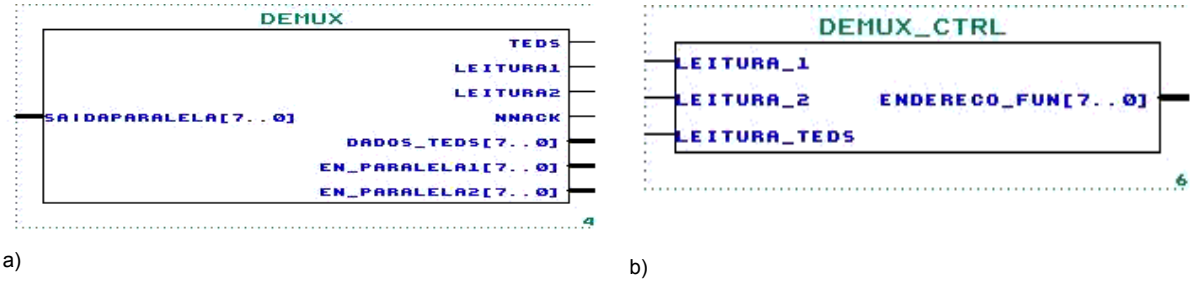
O módulo *gerenciador* é responsável por controlar o fluxo de dados e sinais de controle através da interface TII, do lado do NCAP. O módulo é responsável por obter os endereços de função, canal e escrita fornecidos pelo *software* do NCAP, de forma paralela, e disponibilizá-los para o STIM utilizando o sinal DIN, de forma série, bem como obter os dados que entram via DOUT, de forma série, e disponibilizá-los, de forma paralela, para processamento interno no NCAP.

Os dados que entram através de DOUT são divididos internamente e colocados em três barramentos de oito *bits*, SAIDA_1[7..0], SAIDA_2[7..0] e DADOS_TEDS[7..0], para serem enviados ao módulo demultiplexador. O diagrama do bloco funcional *gerenciador* é apresentado na Figura 7.20.

Figura 7.20. Módulo gerenciador.

e) Módulo Demultiplexador

O módulo demultiplexador (*Demux*) é responsável por obter os dados que entram via EN_PARALELA1[7..0], EN_PARALELA2[7..0] e DADOS_TEDS[7..0] e disponibilizá-los para o barramento de saída SAIDAPARALELA[7..0], do Gerenciador de Protocolo, de acordo com o sinal de controle fornecido pelo bloco *Demux_Ctrl*, a fim de que o *software* do NCAP possa realizar a leitura de dados. Os blocos funcionais são mostrados na Figura 7.21.



a) b)

Figura 7.21. a) Módulo demultiplexador, b) Controle do demultiplexador

7.2.2.2 - Gerenciador de Protocolo com Entrada de Oito *Bits* e Saída de Cinco *Bits*

Esta versão do Gerenciador de Protocolo foi utilizada na implementação final do sistema. O diagrama esquemático é mostrado na Figura 7.22. O dispositivo é composto basicamente pelos seguintes blocos: módulo de entrada de dados da porta paralela, módulo de disparo, detecção do STIM e gerenciador. Os códigos fonte correspondentes são fornecidos no disquete anexado ao final do trabalho.

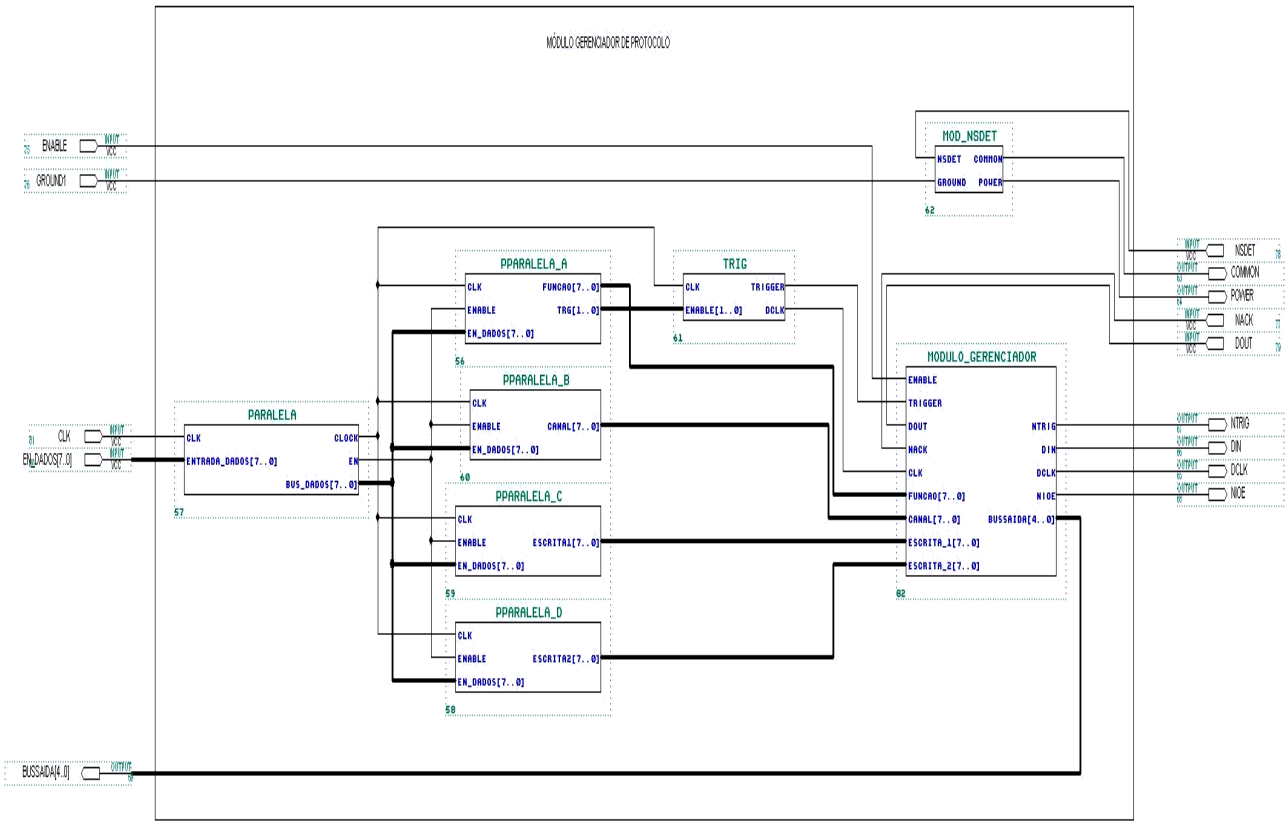


Figura 7.22. Gerenciador de Protocolo com entrada de oito *bits* e saída de cinco *bits*.

Em contraste com o Gerenciador de Protocolo apresentado na seção 7.2.2.1, nesta versão dispõe-se de uma entrada de 8 *bits* de dados sem sinais de controle, pois a multiplexação é feita com endereços de 8 *bits* que circulam pelo mesmo barramento. As operações de leitura são feitas através de uma saída de 5 *bits*, com a finalidade de utilizar o pacote *Parport* diretamente.

De acordo com as recomendações do pacote *Parport*, a leitura de dados deve ser feita através do registrador de estados da porta paralela, usando a configuração mostrada na Tabela 7.1.

Tabela 7.1. Pinagem do registrador de estados da porta paralela para funções de leitura.

Bit	Pino #
7	11 (invertido)
6	10
5	12
4	13
3	15

Internamente, no gerenciador, é lido um *byte* por vez, enviado pelo STIM. Cada *byte* é dividido em duas parcelas, na qual a primeira contém os 5 primeiros *bits*, sendo que o *bit* mais significativo deve ser invertido e, a segunda parcela, contém os 3 *bits* restantes com o *bit* mais significativo invertido; a parcela é preenchida com zeros até completar os 5 *bits*. O *software* do NCAP realiza dois ciclos de leitura por cada *byte* enviado pelo STIM e o *byte* real é reconstruído internamente no NCAP. Na Figura 7.23 ilustra-se o mecanismo de leitura mencionado.

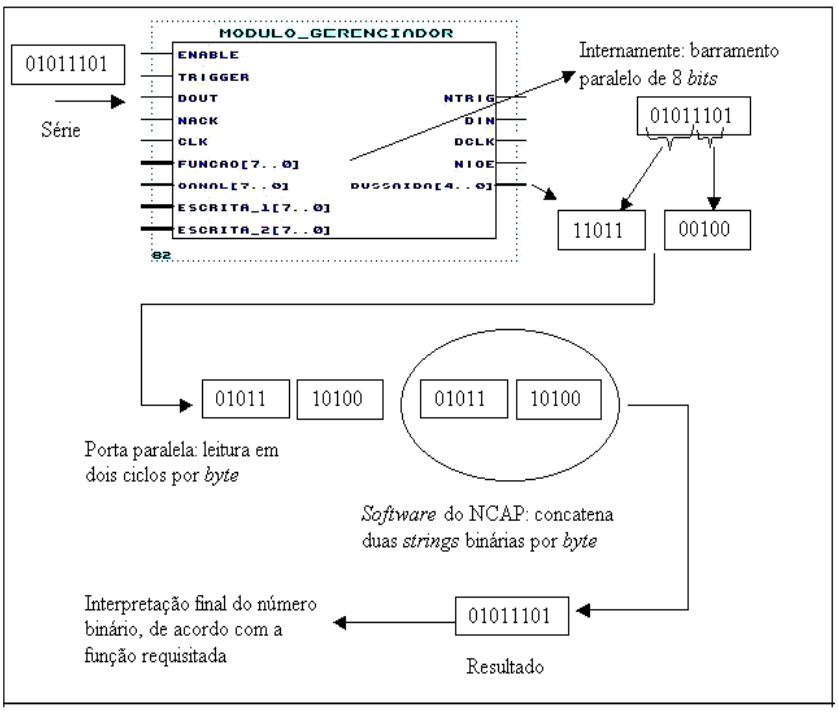


Figura 7.23. Algoritmo para conversão de um *byte* em duas parcelas de 5 *bits*, utilizando dois ciclos de leitura.

Esta versão do Gerenciador de Protocolo também realiza uma multiplexação dos dados que vêm da porta paralela. Porém, com o objetivo de usar apenas o registrador de dados, a multiplexação é feita de acordo com a seqüência mostrada na Figura 7.24. Empregaram-se os endereços 01111111, 11111100, 11111101 e 11111110 para realizar o controle de dados de entrada, pois são valores que, de acordo com o padrão IEEE 1451.2, podem ser utilizados a critério do projetista.

```
EN_DADOS[7..0] =  01111111   => DISPARO
EN_DADOS[7..0]   => FUNÇÃO
EN_DADOS[7..0] = 11111100
EN_DADOS[7..0]  => CANAL[7..0]
EN_DADOS[7..0] = 11111101
EN_DADOS[7..0]   => ESCRITA_1 [7..0]
EN_DADOS[7..0] = 11111110
EN_DADOS[7..0]  => ESCRITA_2 [7..0]
```

Figura 7.24. Multiplexação de dados da porta paralela usando apenas o registrador de dados.

Além das funções já apresentadas na seção 7.2.2.1d, o módulo gerenciador realiza uma demultiplexação interna dos barramentos de dados, além de implementar o algoritmo que transforma um *byte* em duas parcelas de 5 *bits*. O módulo de disparo (*trig*) e de detecção (*mod_nsdet*) desempenham as mesmas funções que já foram comentadas quando foi analisado o Gerenciador de Protocolo com entrada e saída de 8 *bits*.

Como no caso anterior, esta versão foi desenvolvida com base no conceito de máquina de estados, utilizando a descrição VHDL comportamental. Foram necessários 96 estados para atingir o nível de desempenho desejado. Esta solução é compatível com o pacote *Parport* e, portanto, mais conveniente para ser empregada na implementação do nó IEEE 1451 proposto.

7.2.3 - *Software* para o NCAP

Os grupos de pesquisa em Sistemas Digitais e Instrumentação Eletrônica da FE/IS - UNESP têm dedicado esforços na implementação de sistemas envolvendo sensores distribuídos integrados em

rede de comunicação [11], [12], [13], [104]. Batista, em [105], desenvolveu as bases do *software* para o NCAP, que no presente trabalho foi avaliado, testado e modificado com o nó IEEE 1451 completo.

Com o objetivo de dar uma contribuição aos projetos de nós IEEE 1451 baseados em ferramentas abertas, empregou-se a tecnologia Java para desenvolver o *software* do NCAP.

a) Motivação para o Emprego da Tecnologia Java

Escolheu-se a linguagem Java por quatro motivos fundamentais:
a) Utiliza os recursos da orientação a objetos: o padrão IEEE 1451.1 recomenda o emprego de uma linguagem orientada a objeto para descrever as funcionalidades do NCAP.
b) Introduz portabilidade: a linguagem Java é executada em qualquer ambiente que implemente a Máquina Virtual Java.
c) Domínio público: o *Kit* de Desenvolvimento Java (JDK) e as APIs necessárias podem ser obtidas gratuitamente no *site* da *Sun® Microsystems*[36] [88].
d) Possibilita a reutilização de código: Java permite a reutilização de códigos, facilitando a manutenção e a escalabilidade de projetos.

7.2.3.1 - Arquitetura de *Software* usando Java RMI

Os elementos do *software* para o NCAP são compostos por classes que definem a sua funcionalidade. Assim, existem classes para viabilizar a comunicação remota, para a interface gráfica e para o controle do dispositivo Gerenciador de Protocolo.

Através da API RMI é possível o estabelecimento de uma conexão remota baseada no modelo cliente-servidor. Uma estação cliente conectada à rede pode acessar um canal transdutor através da máquina que age como servidor, na qual está implementado o NCAP.

O controle do Gerenciador de Protocolo foi realizado através do pacote de comunicação *javax.comm.* Este pacote contém classes que possibilitam a comunicação com as portas serial e paralela do PC. Junto ao pacote *javax.comm* empregaram-se os pacotes *java.io* e *java.util*, visando controlar o dispositivo conectado ao PC. Um cliente pode acessar uma determinada

[36] Embora a tecnologia JDK pertença à Sun, ela é considerada de domínio público.

aplicação através de uma interface gráfica. Nesta etapa do projeto foi implementada uma interface gráfica simples com base no pacote Java *Swing*.

Para executar um comando remoto, por exemplo, ligar o ventilador, é necessário enviar o endereço de função 16 aplicado ao canal transdutor 2. Isto pode ser feito via interface gráfica. Deste modo, o valor ingressa no Gerenciador de Protocolo, que é responsável por gerenciar o protocolo IEEE 1451.2, iniciando o processo de escrita com um disparo através da linha NTRIG da interface TII. Para desligar o ventilador é necessário enviar um comando de *reset*, endereçando-o para o canal 2. Assim, o *software* desenvolvido oferece a possibilidade de um cliente enviar, de forma remota, o valor de um endereço de função para o NCAP, que age como servidor. Em essência o *software* é composto por três classes principais:

a) *IEEE1451_BaseTransducerBlock*. Esta classe é, na verdade, uma interface Java RMI cujo comportamento é implementado pelo servidor. Vale a pena lembrar que em toda arquitetura RMI, deve existir uma interface remota.

b) *IEEE1451_FunctionBlock*. Esta classe está associada com o lado cliente, fornecendo o IP do cliente ao servidor. Este programa implementa uma interface gráfica simples a fim de que o cliente possa interagir com o servidor, através da rede.

c) *IEEE1451_NCAPBlock*. Constitui o lado servidor, ou seja, o programa que reside no NCAP. Esta classe é responsável pela implementação da interface remota, detecta qual o cliente que está requisitando serviço e possibilita a comunicação com o Gerenciador de Protocolo. O lado cliente pode residir no NCAP, neste caso, deve ser configurado o modo *localhost*, no ambiente RMI.

Salienta-se que, com esta versão do *software* para o NCAP podem ser realizadas operações de escrita, mas não de leitura. O desenvolvimento do *software* para o NCAP baseado em Java RMI, foi a idéia inicial do projeto e com o transcorrer do trabalho viu-se a necessidade de melhorar o desempenho do *software* com o emprego de Java NET, fundamentalmente por questões de velocidade e de comunicação via Internet. O processo completo de escrita-leitura foi desenvolvido levando em consideração um ambiente cliente-servidor baseado em Java NET e o pacote *Parport*, como será analisado em 7.2.3.2.

Voltando, então, ao desenvolvimento do *software* para o NCAP com base no pacote RMI, têm-se, na Figura 7.25, três fluxogramas correspondentes a métodos dentro dos programas cliente e servidor. Cumpre salientar que na programação orientada a objetos é utilizado o diagrama de classes, no entanto, os fluxogramas simplificados são apenas ilustrativos.

Na Figura 7.25a é mostrado o fluxograma do método que implementa o comportamento necessário à escrita de dados, usando a porta paralela e os recursos da API de comunicações. Na Figura 7.25b é apresentado o fluxograma do método que implementa o lado servidor da arquitetura RMI, e na 7.25c, o fluxograma do método que implementa o cliente. Os métodos das Figuras 7.25a e b pertencem ao programa *IEEE1451_NCAPBlock*, que implementa o servidor. Já o fluxograma do método apresentado na Figura 7.25c, pertence ao programa *IEEE1451_FunctionBlock*, que implementa o cliente.

O símbolo correspondente à estrutura de seleção *if*, é na verdade, o tratamento de exceções em Java, através dos comandos *try – catch* (tentativa-captura). Por meio deles é possível declarar uma exceção quando acontece um problema na tentativa de executar uma determinada tarefa.

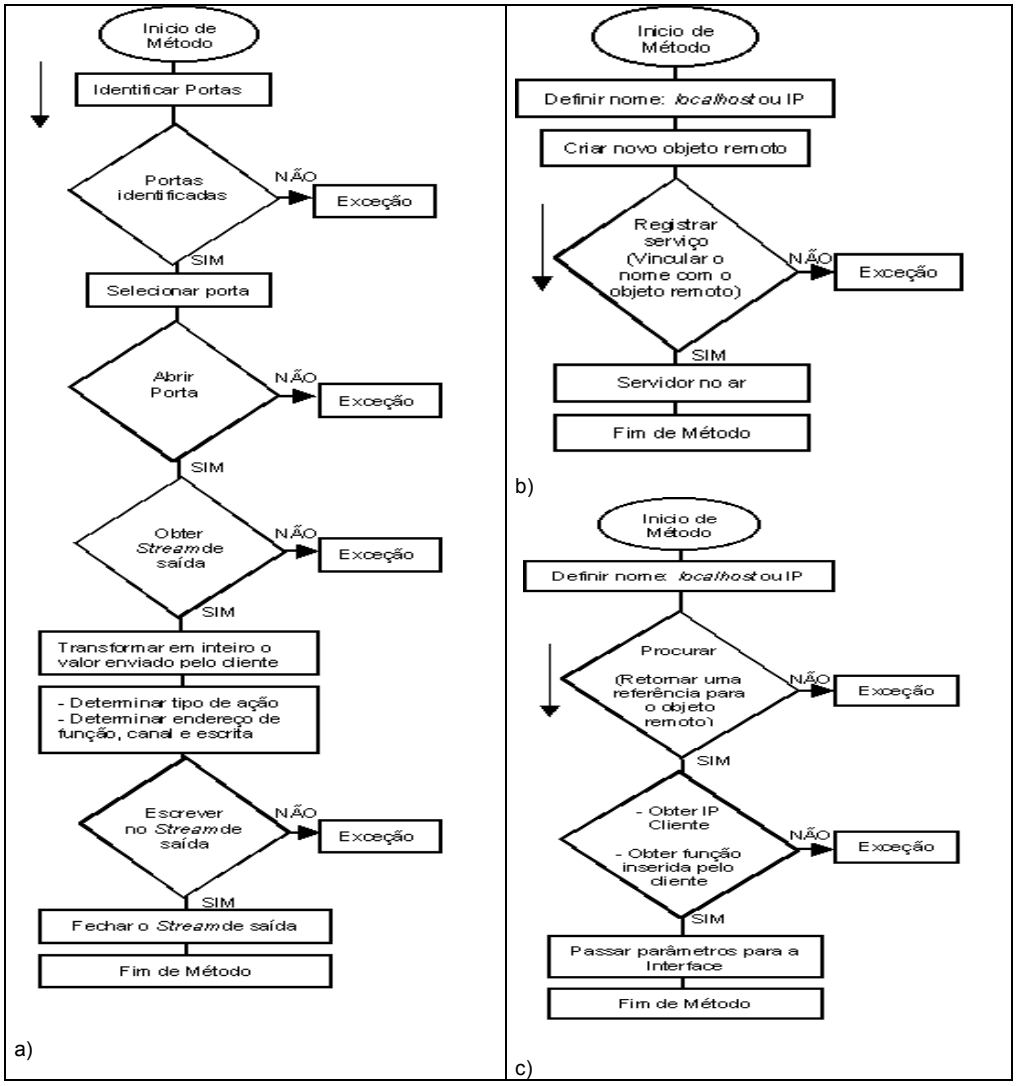


Figura 7.25. Fluxogramas ilustrativos: a) comunicação com a porta paralela; b) servidor; c) cliente.

No diagrama da Figura 7.26 é mostrada a arquitetura de *software* implementada entre o NCAP e um cliente. *A classe IEEE1451_NCAPBlock* herda o comportamento da classe *UnicastRemoteObject*, pertencente ao pacote RMI, e faz uso dos recursos fornecidos por métodos e variáveis das classes *RemoteException, Enumeration, CommportIdentifier, OutputStream, ParallelPort* e *Naming*. A classe *IEEE1451_NCAPBlock* recebe uma coleção de objetos da classe *IEEE1451_FunctionBlock* e implementa a interface *IEEE1415_BaseTrasnducerBlock* que, por sua vez, herda o comportamento da interface *Remote*.

A classe IEEE1451_FunctionBlock faz uso dos recursos fornecidos por métodos e variáveis das classes *Naming, JOptionPane* e *InetAddress*.

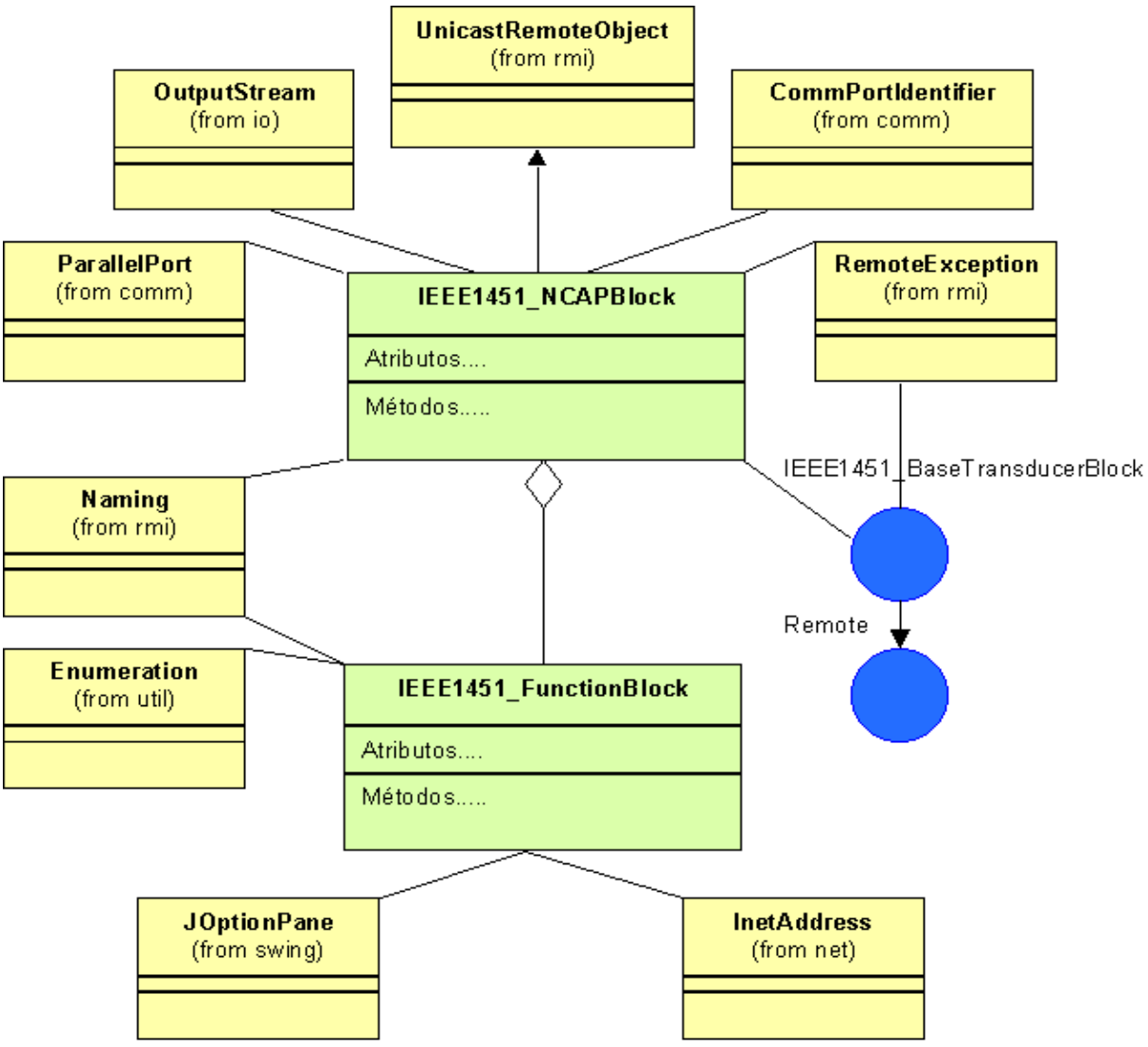


Figura 7.26. Diagrama da arquitetura *software* do NCAP, com base em Java RMI.

7.2.3.2 - Arquitetura de *Software* Baseada em Java NET

Através da API NET é possível o estabelecimento de uma conexão remota cliente-servidor com base no conceito de *sockets*. Como acontece com RMI, uma estação cliente conectada à rede pode acessar um canal transdutor através da máquina que age como servidor, na qual está implementado o NCAP, entretanto, a API NET é mais eficiente quando se estabelecem conexões via Internet. Por outro lado, embora seja possível o emprego de RMI em comunicações via Internet, seu funcionamento pode ser barrado pela ação dos *firewalls*[37].

Assim sendo, a comunicação entre cliente e servidor se estabelece diretamente por meio de *sockets*. O servidor acessa os canais transdutores do STIM através da intermediação do Gerenciador de Protocolo com entrada de 8 *bits* e saída de 5 *bits*, utilizando a porta paralela do PC. Neste nível, a comunicação é estabelecida graças à ação do pacote *Parport*, introduzindo uma melhora notável na velocidade e eficiência do sistema.

Embora a portabilidade do *software* possa ver-se comprometida pelo emprego de métodos nativos em nível de comunicação com a porta paralela do PC, as possíveis reconfigurações de um sistema operacional para outro são mínimas, pois a instalação do pacote *Parport* é muito simples.

No diagrama da Figura 7.27 é apresentada a arquitetura de *software* implementada entre o NCAP e um cliente, usando Java NET.

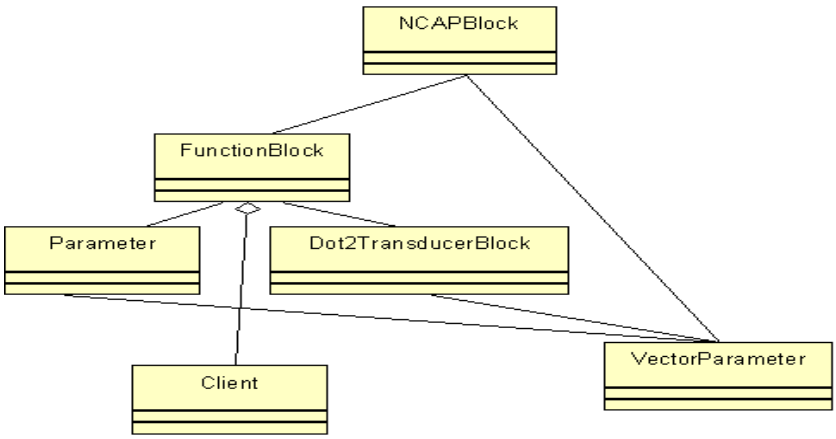


Figura 7.27. Diagrama simplificado da arquitetura *software* do NCAP, com base em Java NET.

O software do NCAP foi concebido basicamente com base nas seguintes APIs:

[37] *Firewall*: Barreira de segurança baseada em *hardware* e *software* que pode rodar em um microcomputador e cuja finalidade é proteger uma rede corporativa contra acessos externos indesejáveis.

a) NET: com a finalidade de criar uma interação cliente-servidor através de *sockets* e não re-configurar a aplicação quando se precise migrar da rede local para a Internet.
b) *io*: com o objetivo de receber e enviar informações através da rede usando fluxos de entrada e saída (*streams*).
c) AWT *(Abstract Window Toolkit*): para a criação das interfaces gráficas dos aplicativos cliente e servidor.
d) *Parport*: para escrever dados no registrador de dados da porta paralela e ler dados do registrador de estados, de uma maneira eficiente.

De acordo com o diagrama da Figura 7.27 foram implementadas seis classes diferentes, sendo que cada uma delas implementa interfaces ou herda comportamentos de outras classes da API Java. Em essência o *software* é composto pelas seguintes classes:
a) *NCAPBlock*: esta classe contém o método principal e, através dela, é executado o servidor na classe *FunctionBlock*. Ao ser executada, a classe *NCAPBlock* carrega as TEDS associadas com o STIM que o NCAP tiver conectado nesse momento. As informações TEDS são armazenadas em arquivos de texto. Quando um aplicativo cliente requisitar a leitura das TEDS, o aplicativo lerá o conteúdo desses arquivos, pois as informações das TEDS são estáticas e só mudam se o módulo STIM for trocado por outro, bastando apenas, rodar novamente o programa *NCAPBlock* sem realizar qualquer reconfiguração para carregar as novas informações TEDS.

No diagrama da Figura 7.28 é mostrada a classe *NCAPBlock* com seus principais atributos e seus métodos, e as classes e recursos utilizados da API Java.

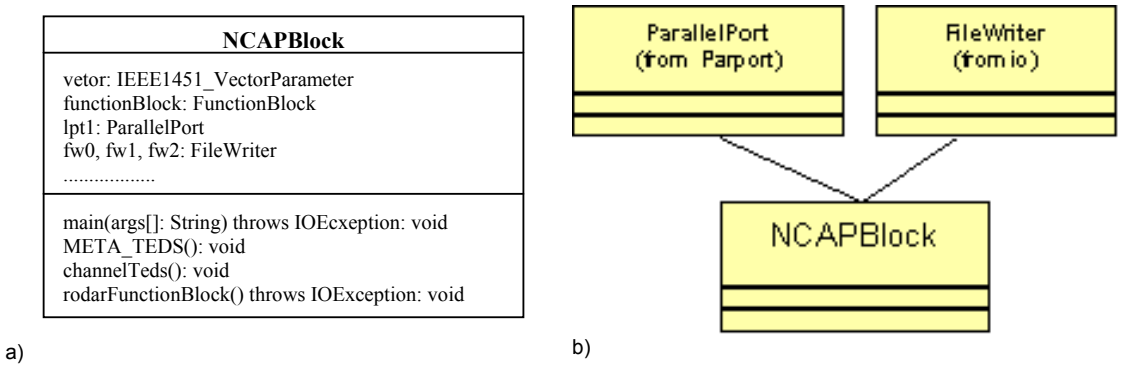


Figura 7.28. Classe *NCAPBlock*: a) Diagrama de classe, b) recursos usados da API Java.

b) *FunctionBlock*: esta classe implementa o lado servidor da aplicação e faz o encapsulamento de uma parte da funcionalidade do NCAP. O diagrama desta classe é mostrado na Figura 7.29.

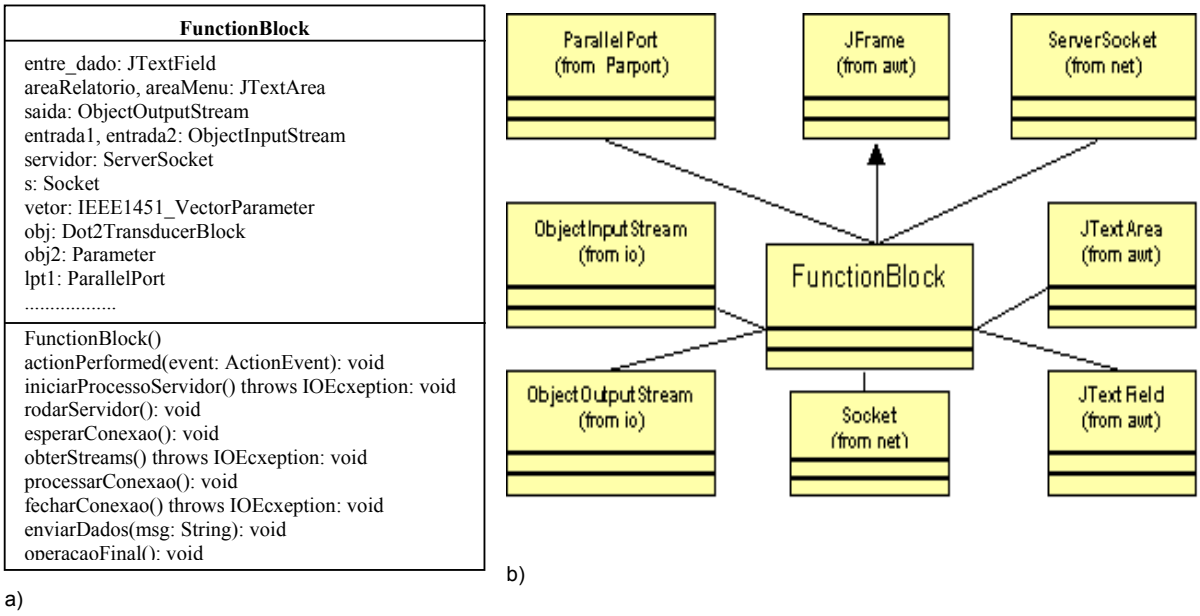


Figura 7.29. Classe *FunctionBlock*: a) Diagrama de classe, b) recursos usados da API Java.

Através da classe *FunctionBlock* é criado um *socket*, a fim de que um cliente possa estabelecer conexão com o servidor, posteriormente o objeto denominado *s*, da classe *Socket*, interage com os fluxos de entrada e saída da classe *io*, com a finalidade de transmitir e receber dados utilizando a rede local ou a Internet. O trecho de código fonte da classe *FunctionBlock* da Figura 7.30, ilustra o conceito. Note-se que o *socket* deve ser criado em uma porta TCP não reservada.

```
public void rodarServidor() {
   try { //tentativa
       //1º: Criar um socket no servidor
      servidor = new ServerSocket (8780, 100);
       -
private void obterStreams() throws IOException {
       //Configurar o fluxo de saída para objetos
    saida = new ObjectOutputStream( s.getOutputStream() );
         //Configurar o fluxo de entrada para objetos
    entrada1 = new ObjectInputStream( s.getInputStream() );
    entrada2 = new ObjectInputStream( s.getInputStream() );
     -
```

Figura 7.30. Trecho de código da classe *FunctionBlock*.

A classe *FunctionBlock* emprega a classe *ParallelPort* pertencente ao pacote *Parport*, para realizar operações de escrita e leitura usando a porta paralela e a funcionalidade do IEEE 1451.2.

c) *Client*: esta classe implementa o lado cliente da aplicação e pode rodar no modo *localhost* ou remoto, através da rede local ou da Internet, bastando apenas a configuração do IP do servidor. O diagrama da classe *FunctionBlock* é apresentado na Figura 7.31. A classe emprega vários recursos da API AWT, a fim de implementar uma interface gráfica simples e amigável.

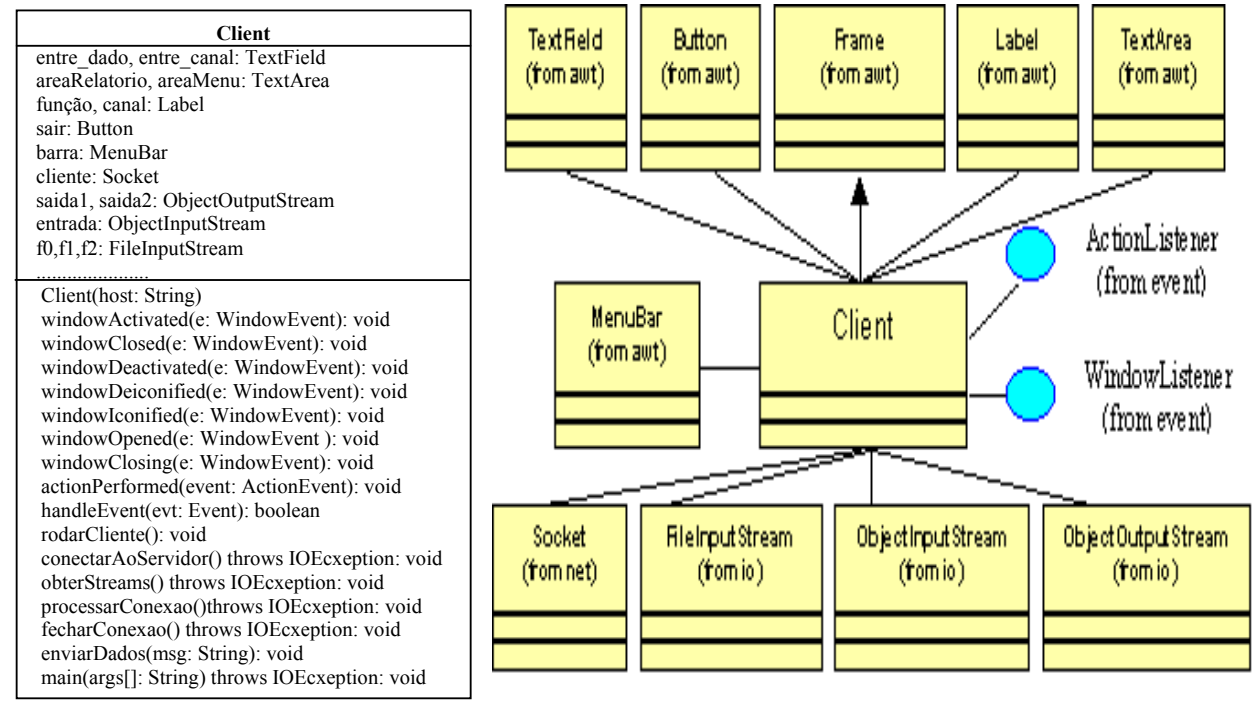


Figura 7.31. Classe *Client*: a) Diagrama de classe, b) recursos usados da API Java.

d) *Dot2TransducerBlock, Parameter e VectorParameter*: as instâncias destas classes são utilizadas pelas classes *NCAPBlock* e *FunctionBlock* para realizar o tratamento de dados. Na classe *Dot2TransducerBlock* se realiza o tratamento de dados associados com um canal transdutor do STIM. O dado lido através da porta paralela e interpretado inicialmente como uma *string* binária e, logo depois, deve ser tratado e interpretado de uma maneira conveniente a fim de que o usuário possa interpretá-lo. Por exemplo, como uma palavra, um número inteiro ou um

número de ponto flutuante de acordo com o padrão IEEE 754. A classe *Parameter* realiza este tratamento no caso das leituras estarem relacionadas com os registradores de estado, dados e interrupção. A classe *VectorParameter* é responsável por realizar dois ciclos de leitura por cada *byte* enviado pelo STIM e interpretá-los de maneira correta, como mencionado em 7.2.2.2. Os diagramas correspondentes a estas classes são mostrados na Figura 7.32a, b e c.

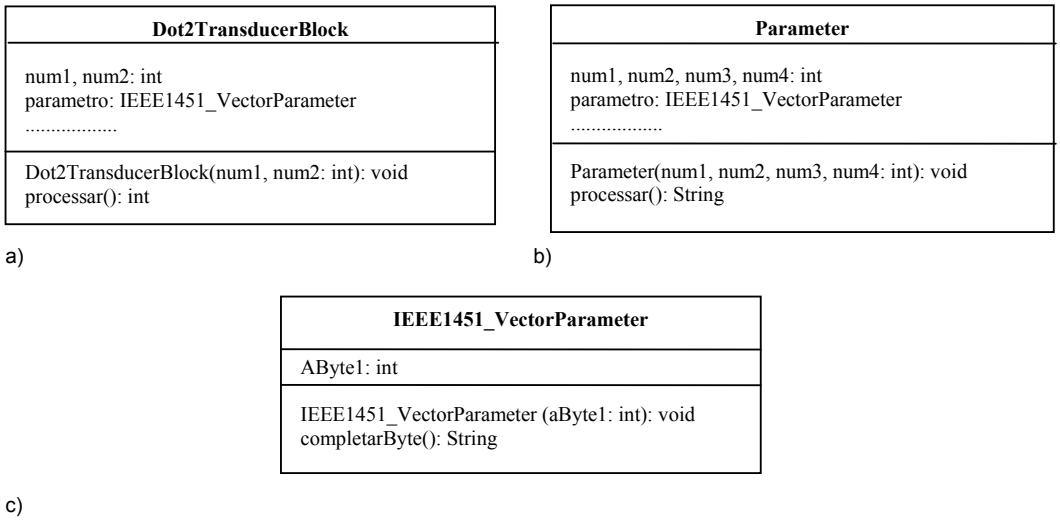


Figura 7.32. Classes: a) *Dot2TransducerBlock*, b) *Parameter* e c) *VectorParameter*.

Cumpre salientar que os códigos fonte correspondentes são fornecidos no disquete anexado ao final do trabalho.

7.3 - Conexão do Nó IEEE 1451 na Rede

O nó IEEE 1451 (STIM-NCAP) foi conectado em rede, em um ambiente distribuído, sob comunicação cliente-servidor. O NCAP– PC foi conectado a uma rede *Ethernet* 10 BASE T, sendo que uma, ou várias, aplicações clientes podem interagir com o NCAP através da rede. O modelo de comunicação em rede empregado é apresentado na Figura 7.33a e na Figura 7.33b, ilustra-se a interação cliente-servidor.

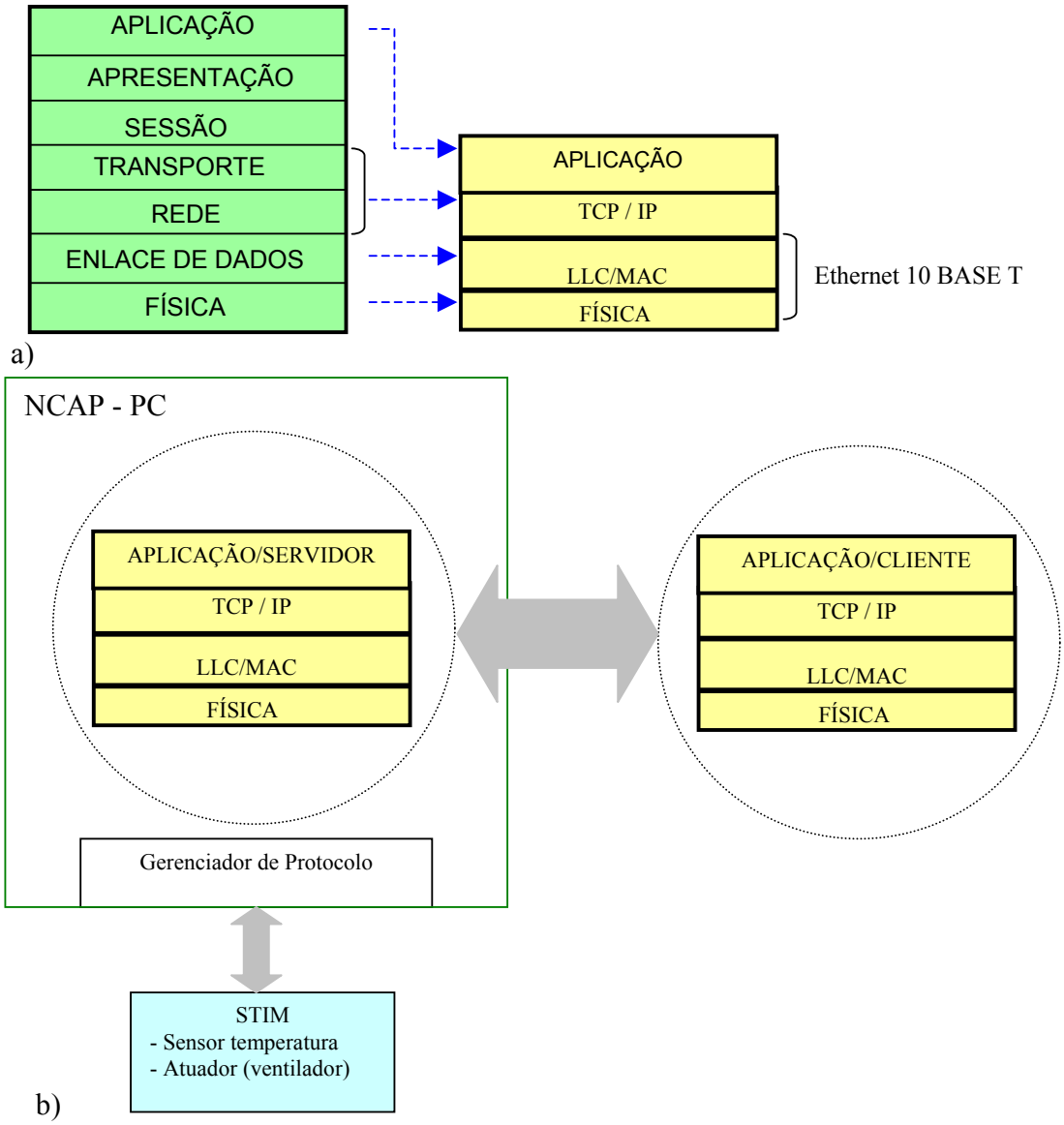


Figura 7.33. Conexão do nó IEEE 1451 na rede. a) modelo de comunicação; b) interação cliente-servidor.

7.4 – Comentários Finais sobre o Capítulo 7

Neste capítulo foi realizada uma descrição dos materiais necessários e de como as tecnologias apresentadas no Capítulo 6 foram usadas a fim de atingir os objetivos do projeto. No Capítulo seguinte serão expostos os resultados obtidos. As simulações, resultados experimentais e testes de laboratório realizados têm como objetivo esclarecer os conceitos apresentados neste Capítulo.

Capítulo 8

Resultados Obtidos

Nesta seção são expostos os resultados do trabalho, tanto de simulações quanto experimentais, obtidos em laboratório. Serão apresentados diversos resultados associados com o STIM, com o Gerenciador de Protocolo e com o *software* do NCAP.

8.1 - Implementação do STIM

O STIM foi implementado através de duas alternativas diferentes. Na primeira delas, o módulo de controle da TII foi projetado como sendo um sistema assíncrono e, na segunda, como sendo um sistema síncrono.

a) Alternativa 1 – Sistema Assíncrono

Inicialmente foram tomados como base os sinais NTRIG, NACK e NIOE assíncronos. Os processos internos do STIM foram sincronizados com base no sinal DCLK.

Parte do módulo de controle da TII foi descrito por meio do conceito de máquina de estados. Entretanto, foram empregados contadores para realizar as temporizações dos sinais e a contagem de *bits*. Inicialmente, o projeto foi sintetizado levando em conta apenas um canal, e implementado com uma FPGA de uso geral da família FLEX 10K modelo EPF10K20RC240-4.

Posteriormente, o projeto foi sintetizado levando em conta 2 e 3 canais, respectivamente, e implementado com FPGAs da família ACEX, modelos EP1K50TC144-3 e EP1K100QC208-3.

Quando testado de forma isolada, o STIM exibiu comportamento satisfatório. Porém, quando conectado ao Gerenciador de Protocolo foram constatados problemas de sincronismo na transferência de dados, que se atribuem ao emprego de contadores para realizar os processos de contagens de *bits*. Esta metodologia aumenta drasticamente o número de recursos internos necessários, além de apresentar deficiências quando usada como suporte na implementação de protocolos. A solução para este tipo de problemas é o uso do conceito de máquina de estados.

b) Alternativa 2 – Sistema Síncrono

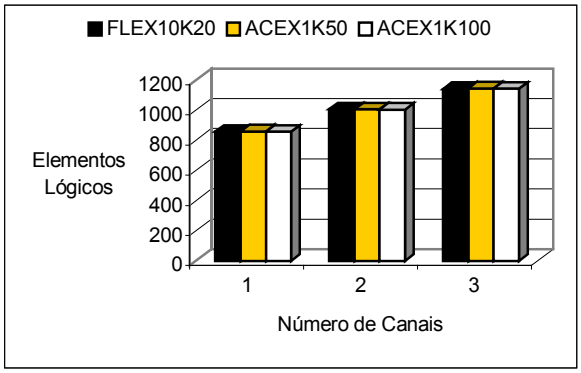
Neste caso, o módulo de controle da TII foi concebido integralmente com base no conceito de máquina de estados. Utilizou-se o sinal POWER como *reset* geral do sistema e todos os demais sinais foram sincronizados com o sinal DCLK. Eliminou-se, assim, o emprego de contadores no módulo de controle da TII e o gerenciamento passou a ser feito através de estados.

O projeto foi sintetizado levando em conta apenas um canal, e implementado com uma FPGA de uso geral da família FLEX 10K modelo EPF10K20RC240-4. Posteriormente, o projeto foi sintetizado levando em conta 2 e 3 canais, respectivamente, e implementado com FPGAs da família ACEX1K, modelos EP1K50TC144-3 e EP1K100QC208-3.

Quando testado de forma isolada e conectado ao Gerenciador de Protocolo, o STIM exibiu comportamento satisfatório. Constatou-se que o sistema concebido de acordo com a segunda alternativa diminui consideravelmente o número de células lógicas utilizadas para sintetizar o projeto. Este fato é ilustrado na Figura 8.1.

Alternativa 1

Sistema Assíncrono - Elementos Lógicos			
Nº de Canais / FPGA	1	2	3
FLEX10K20	855	1004	1139
ACEX1K50	857	1005	1143
ACEX1K100	855	1002	1141



Alternativa 2

Sistema Síncrono - Elementos Lógicos			
Nº de Canais / FPGA	1	2	3
FLEX10K20	555	697	827
ACEX1K50	555	698	830
ACEX1K100	555	699	820

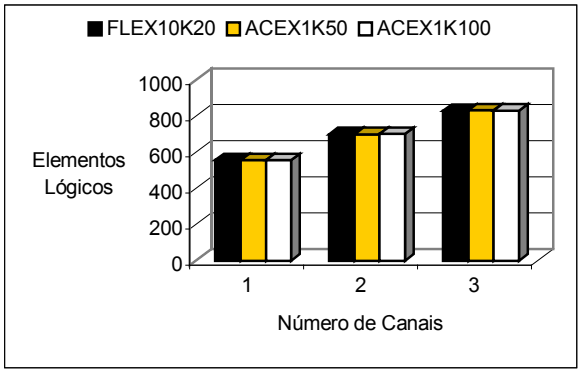


Figura 8.1. Elementos Lógicos em função do tipo de FPGA e do número de canais.

Pode-se observar, na Figura 8.1, que através da segunda alternativa, o número de elementos lógicos utilizados diminui em aproximadamente 300, em todos os casos.

Constatou-se que na primeira alternativa, o fator que produziu o aumento significativo do número de elementos lógicos utilizados foi o emprego de contadores.

8.1.1 - Influência do Número de Canais sobre os Recursos Utilizados

A seguinte análise tem como objetivo ilustrar quais são os principais fatores na escolha de um PLD para implementar um STIM com um dado número de canais transdutores. Embora a análise tenha sido realizada com dispositivos específicos da empresa *Altera*, o conceito é válido para entender como a capacidade do STIM influencia na escolha do PLD para implementá-lo.

A funcionalidade do STIM, levando em consideração o módulo central, os blocos TEDS, os registradores e a interface com os transdutores, para apenas um canal, constitui a implementação básica a partir da qual pode-se incrementar o número de canais transdutores.

Ao ser inserido um novo canal no STIM, o número de elementos lógicos necessários aumenta porque por cada canal acrescentado é preciso adicionar um registrador padrão de estados, um registrador auxiliar de estados, um registrador de interrupções, um bloco de memória Canal-TEDS e um registrador de dados que faz parte do novo canal transdutor.

Por conseguinte, tomando como base a implementação do primeiro canal transdutor são necessários em torno de 560 elementos lógicos através da alternativa dois. A partir desse valor, são necessários em torno de 150 elementos lógicos por cada canal acrescentado, como pode ser analisado na Figura 8.1.

Na Figura 8.2 mostra-se o aumento percentual do número de elementos lógicos em função do incremento de canais. No primeiro gráfico pode se observar que um STIM implementado através da alternativa dois, com três canais, por exemplo: dois canais transdutores e o canal zero, em uma FPGA da família FLEX, ocupa quase a totalidade dos elementos lógicos, ao passo que em uma FPGA da família ACEX, ocupa aproximadamente 70% dos elementos lógicos.

Os outros fatores a serem levados em consideração na implementação de um STIM com dispositivos lógicos programáveis são: a capacidade de memória para implementar as TEDS e o número de entradas e saídas disponíveis, que serão analisados a seguir.

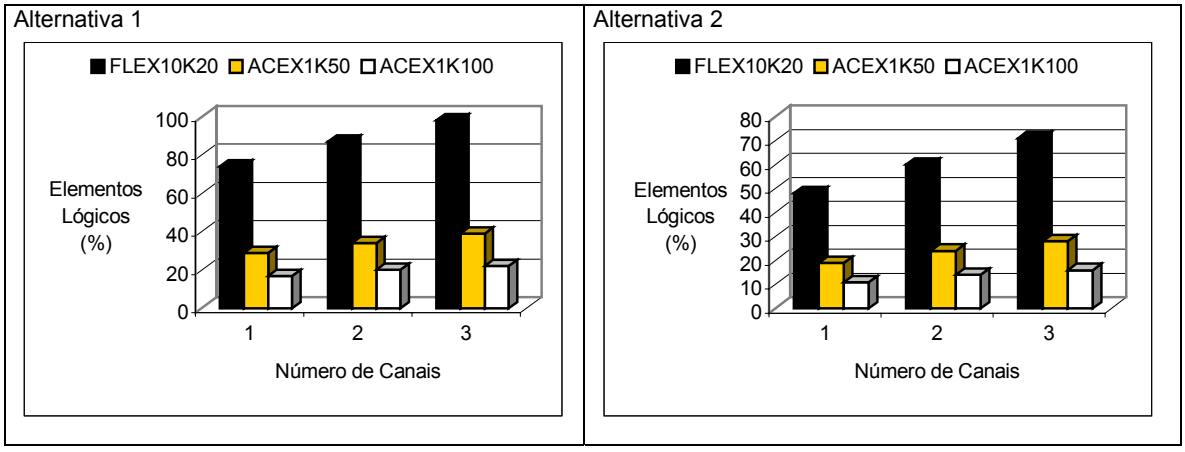


Figura 8.2. Aumento percentual do número de elementos lógicos vs. o incremento de canais.

O número de *bits* necessários para alocar as TEDS aumenta de maneira proporcional com o aumento do número de canais pois, por cada canal acrescentado são necessários 1024 *bits* a mais. Este fato é ilustrado na Figura 8.3.

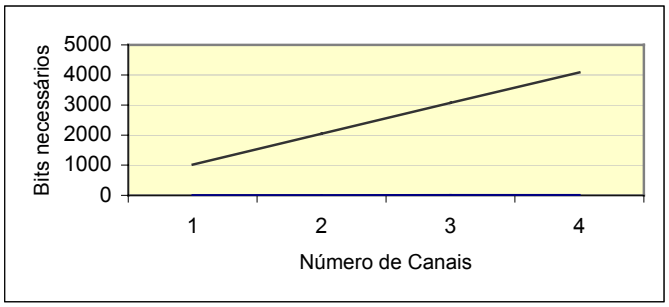


Figura 8.3. *Bits* necessários para alocar as TEDS vs. número de canais.

Quanto ao número de entradas e saídas necessárias deve-se considerar que, além das entradas e saídas projetadas, o STIM precisa de dez sinais para a TII. O valor total deve ser confrontado com o valor disponível no FPGA escolhido.

Para a alternativa 2, escolhida para a implementação definitiva com três canais (2 canais mais o canal 0), utilizando-se uma FPGA ACEX EPF1K50TC144-3, com um canal transdutor operando com 8 *bits* e um canal composto por uma saída digital, obtiveram-se os valores apresentados na Tabela 8.1, onde também são mostrados os recursos utilizados para a mesma implementação com uma FPGA FLEX EPF10K20RC240-4.

Tabela 8.1. STIM; recursos utilizados.

EPF1K50TC144-3			
	Elementos Lógicos	Entradas/Saídas	Bits de Memória
Recursos Disponíveis	2880	102	40960
Recursos Utilizados	830 (28,8%)	19 (18,6%)	3072 (7,5%)
EPF10K20RC240-4			
	Elementos Lógicos	Entradas/Saídas	Bits de Memória
Recursos Disponíveis	1152	189	12288
Recursos Utilizados	827 (71%)	19 (10%)	3072 (25%)

Por exemplo, se fossem implementados 10 canais transdutores mais o canal 0, com 8 *bits* por canal e, considerando que, de acordo com os resultados apresentados na Figura 8.1, para cada canal acrescentado precisam-se em torno de 150 elementos lógicos a mais, se obteriam: 2030 elementos lógicos, 98 I/O e 11264 *bits* de memória. Desta maneira, pode-se notar que no caso do ACEX, o limitante é o número de entradas/saídas, enquanto que no FLEX, o limitante é o número de elementos lógicos e de *bits* de memória disponíveis, em função de sua capacidade menor.

Conclui-se que, para uma aplicação dedicada, com dois canais, o projeto pode ser implementado com uma FPGA FLEX EPF10K20RC240-4. Entretanto, para implementar mais canais transdutores é necessário empregar uma FPGA de maior número de elementos lógicos, como é o caso das FPGAS da família ACEX. Uma aplicação com 10 canais transdutores precisa em torno de 2000 elementos lógicos, sendo factível sua implementação com um dispositivo ACEX EPF1K50TC144-3. Este dispositivo constitui uma solução versátil e de aplicação geral, se comparado com outros PLDs existentes no mercado [29], [101].

8.1.2 - Resultados de Simulações e Experimentais

Na Figura 8.4 são apresentadas diversas etapas da simulação do protocolo de disparo para o sistema síncrono, em um diagrama de tempos. Na Figura 8.4a observa-se que, estando a alimentação inativa, mesmo com um sinal de habilitação por parte do NCAP, o sinal NACK permanece em estado lógico ‘0’. Na Figura 8.4b pode-se observar que quando é fornecida a alimentação através do sinal POWER, e o STIM recebe sinal de habilitação, a linha NACK muda para ‘1’ no próximo estado. Nessas condições, o STIM aguardará um evento de disparo por parte do NCAP, para agir em conseqüência. Quando um evento de disparo acontece, indicado com o número 1 na Figura 8.4c, o STIM reconhece o evento, negando o sinal NACK (2). Quando o

NCAP muda o estado lógico do sinal NTRIG, indicado com o número 3 na Figura 8.4d, o STIM muda o estado lógico do sinal NACK (4), após aguardar um tempo igual a três pulsos de relógio. Na seqüência, quando o STIM recebe o sinal de habilitação através da linha NIOE, o STIM nega o sinal NACK (5), após aguardar dois pulsos de relógio. Nessas condições é possível transferir dados em sincronismo com o sinal DCLK. Depois de reconhecer um evento de disparo, o sinal NACK é usado pelo STIM para delimitar pacotes de 8 *bits* (6) e (7). Este fato é ilustrado na Figura 8.5a, onde também é possível visualizar as etapas já explicadas. Na Figura 8.5b é mostrado o resultado experimental correspondente aos sinais NTRIG e NACK.

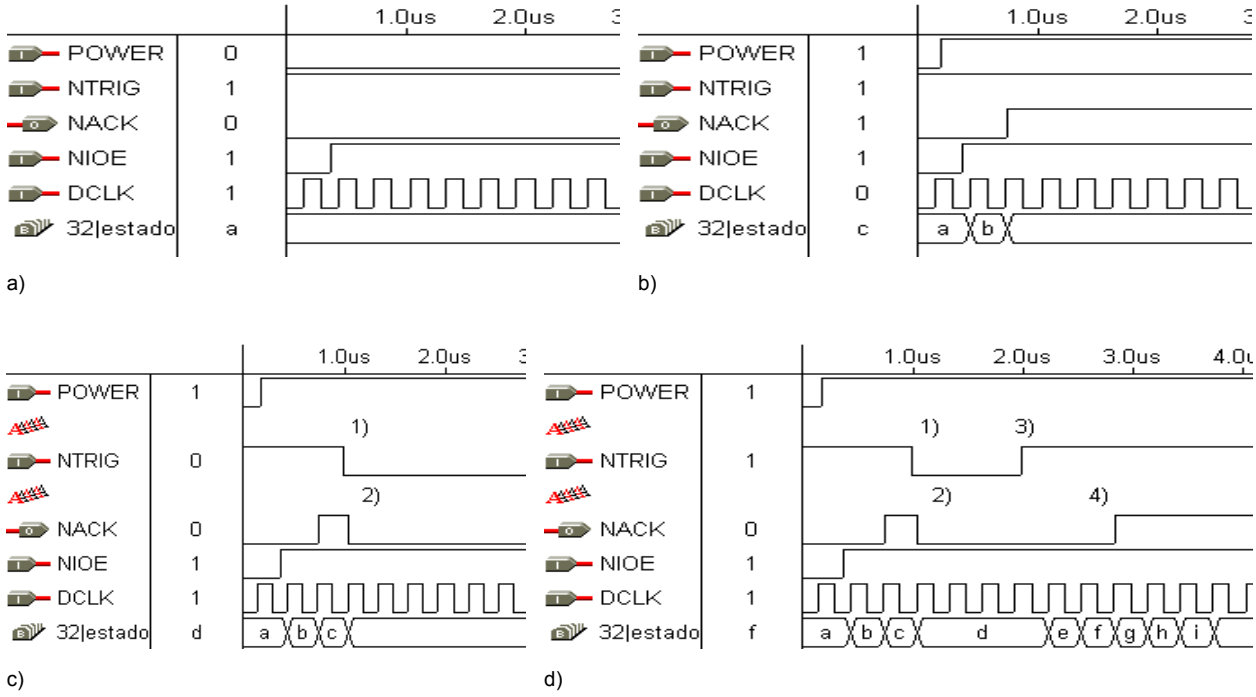


Figura 8.4. Simulação do protocolo de disparo.

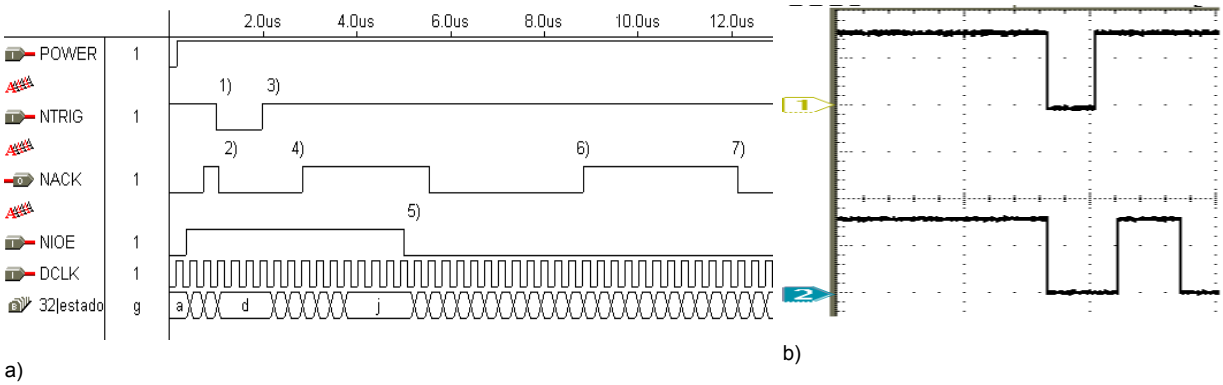


Figura 8.5. a) Protocolo de transferência de bits, b) Disparo; resultado experimental.

8.1.2.1 - Protocolo IEEE 1451.2

Na Figura 8.6a pode se analisar um exemplo de simulação do protocolo IEEE 1451.2, onde as etapas 1 a 7 já foram explicadas. Quando a transferência de um quadro de leitura ou escrita acaba, o NCAP desabilita o STIM através do sinal NIOE (8) e, logo depois, o STIM muda o estado lógico do sinal NACK (9), ficando à espera de um novo processo. Os resultados experimentais correspondentes podem ser observados nas Figuras 8.6b, onde é mostrado o sinal NTRIG vs. O sinal NACK, e 8.6c, onde é mostrado o sinal NIOE vs.o sinal NACK.

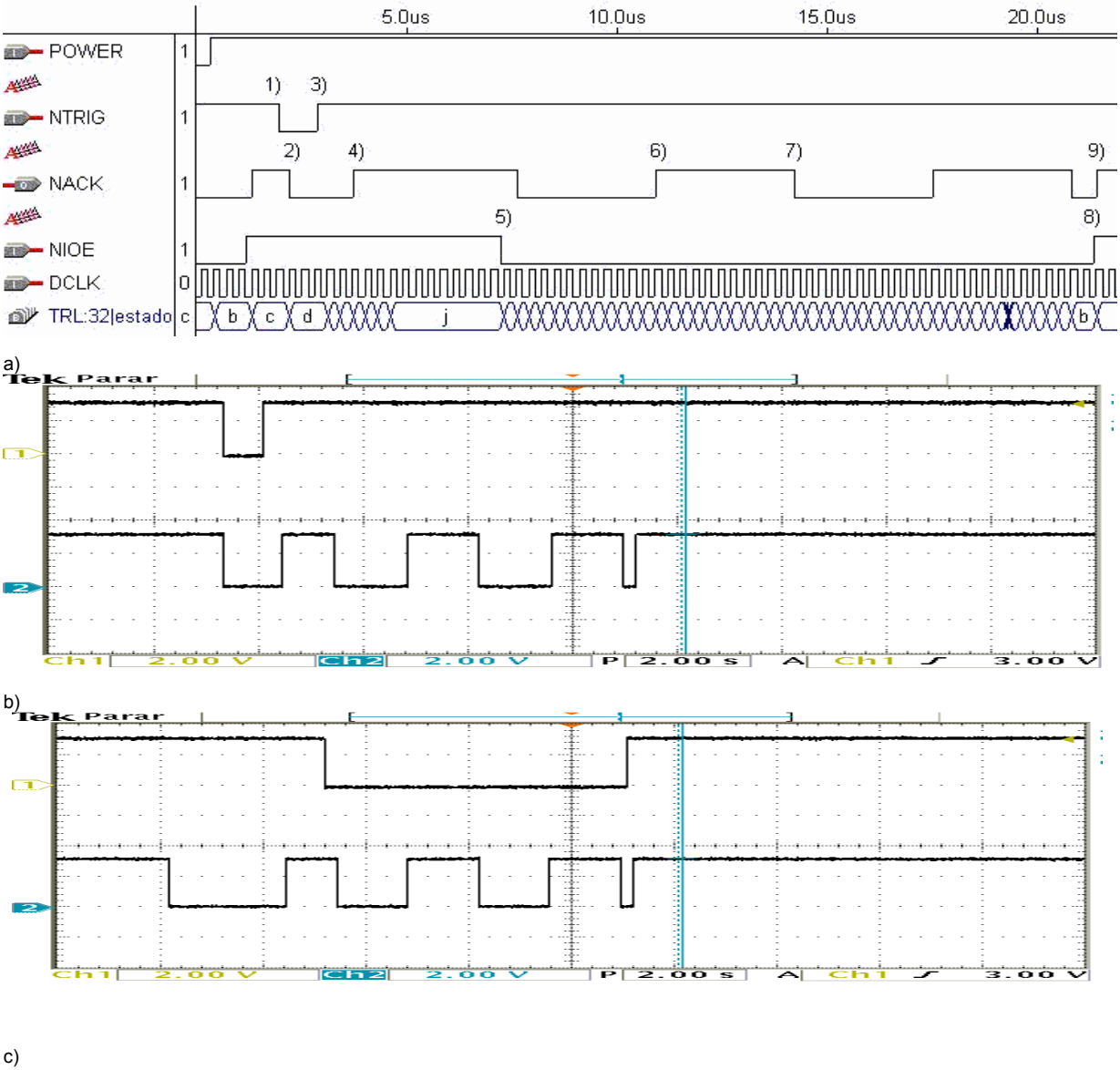


Figura 8.6. Exemplo de protocolo IEEE1451.2: a) simulação; b) e c) resultados experimentais.

No trabalho foram considerados os endereços de função indicados na Tabela 8.2, levando em conta os endereços mais comuns[38] e outros implementados de acordo com as necessidades do projeto.

Tabela 8.2. Endereços de Função utilizados.

Endereço de Função	Descrição
1	Escrever comando de controle
5	Escrever registradores de máscaras de interrupção
16	Executar ação relacionada com um atuador
128	Ler dados de um canal transdutor
130	Ler registrador padrão de estados
132	Ler registrador auxiliar de estados
160	Ler dados das TEDS
240	Ler registradores de interrupção

8.1.2.2 - Lendo Dados do Canal Transdutor Número 1

Um exemplo de transferência de dados, processados pelo STIM, e enviados e recebidos através da interface TII, pode ser observado na Figura 8.7. O exemplo refere-se à leitura dos dados fornecidos pelo canal transdutor número 1.

O primeiro *byte* enviado pelo NCAP através de DIN é o endereço de função, no caso o código "128_{10}" ou "10000000_2", que corresponde à leitura de canal transdutor. O segundo *byte* enviado para o STIM é o endereço de canal, no caso, o código correspondente ao canal transdutor número 1. Como mostrado no exemplo, esses códigos são colocados seqüencialmente em barramentos de 8 *bits* para processamento interno no STIM[39]. Os dados do transdutor, que entram através do barramento denominado ENDADOS[7..0], saem através do sinal DOUT, de forma série, e são enviados para o NCAP. O dado transmitido é "01101010_2" ou "106_{10}".

Após um *byte* ter sido transferido, o STIM aguarda o tempo correspondente a um estado para delimitar o frame através do sinal NACK. Este é o tempo de *hold-off*, ou seja, o tempo que o

[38] Existem 255 endereços de função possíveis, no entanto, a maior parte deles são reservados para futuras modificações do padrão, ou abertos para a indústria, para serem utilizados de acordo com as suas necessidades.
[39] Os barramentos de função, canal e escrita, mostrados no exemplo são internos, não constituem, portanto, entradas e saídas do STIM. Os barramentos internos são mostrados apenas a título ilustrativo.

STIM pode esperar no estado passivo, após finalizar uma transferência de um quadro de leitura ou de escrita. Isto é válido tanto para as transições positivas e negativas do sinal NACK.

Os *bytes* correspondentes a endereço de função, canal e escrita, entram de maneira sincronizada através de DIN, graças à ação do Gerenciador de Protocolo, que faz parte do NCAP.

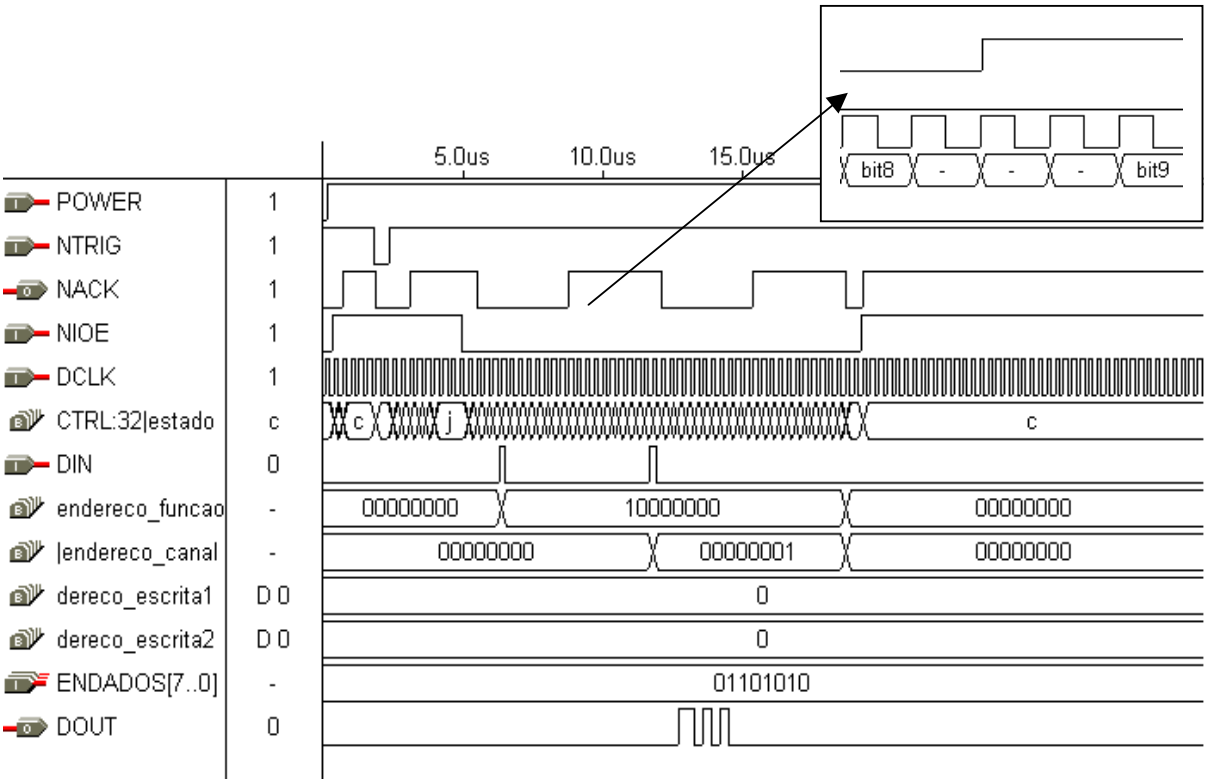
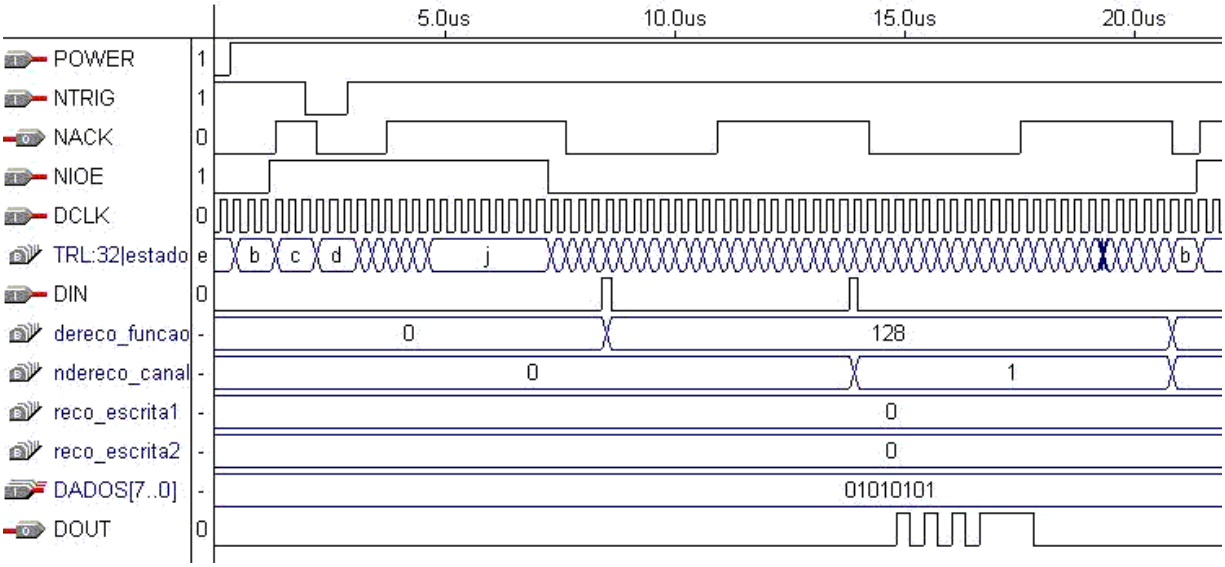


Figura 8.7. Leitura de dados de um canal transdutor.

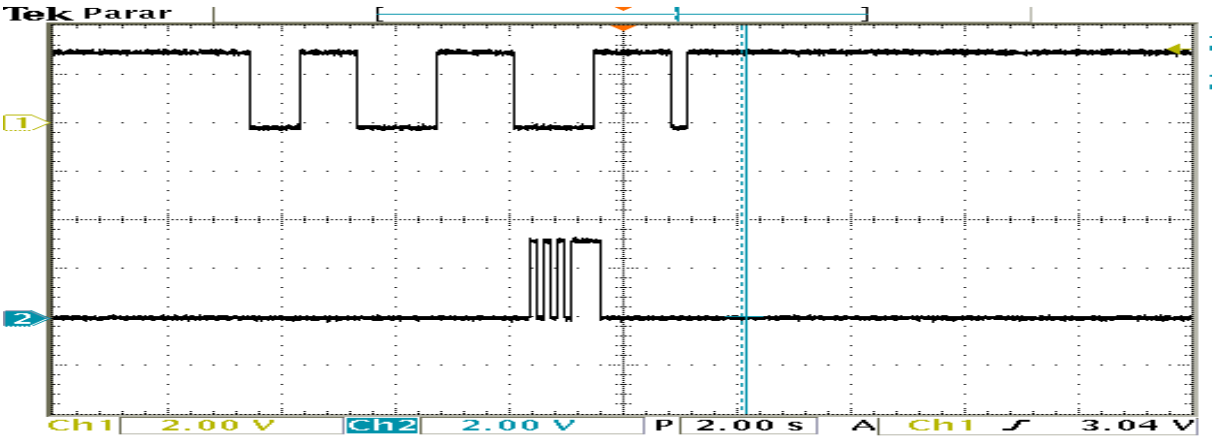
Na Figura 8.8 é mostrado outro exemplo de leitura de dados com o correspondente resultado experimental. O primeiro *byte* enviado pelo NCAP através de DIN é o endereço de função ("128_{10}") e o segundo *byte* enviado para o STIM é o endereço de canal ("1_{10}"). O dado transmitido via DOUT corresponde ao valor binário "01010101_2".

No padrão IEEE 1451.2 não é definido um valor específico de freqüência. Nos casos apresentados, a freqüência do relógio é de 3 MHz[40]. Porém, na prática, foi constatado comportamento satisfatório com freqüências maiores. O valor de freqüência limite através da TII é imposto pela FPGA, como será analisado na seção 8.4.

[40] Valor estimado com base na experiência de Girerd et al.[26].



a)



b)

Figura 8.8. Exemplo de leitura de dados: a) simulação; b) resultado experimental.

8.1.2.3 - Lendo o Registrador Padrão de Estados do Canal 1

O exemplo mostrado no diagrama de tempos da Figura 8.9 refere-se à leitura dos dados contidos no registrador padrão de estados do canal transdutor número 1. O endereço de função enviado pelo NCAP através de DIN é "130_{10}" ou "10000010_2", que corresponde à leitura de registrador padrão de estados. Os dados correspondentes saem através do sinal DOUT e são enviados para o NCAP.

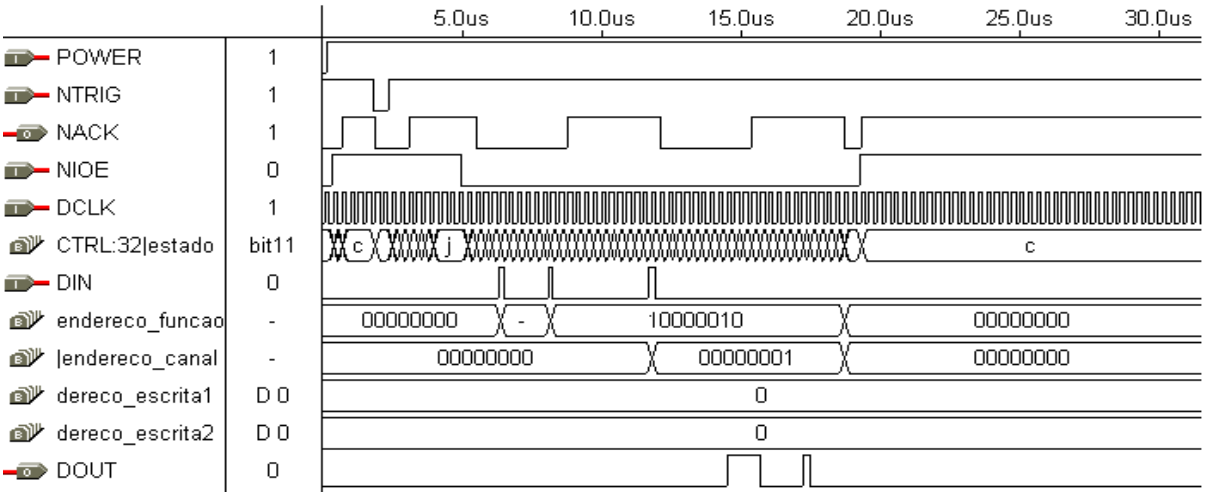


Figura 8.9. Leitura de registrador padrão de estados.

8.1.2.4 - Lendo o Bloco Canal-TEDS

No exemplo seguinte, apresentado na Figura 8.10, é mostrado o processo de leitura dos dados contidos na estrutura Canal-TEDS1. Estes dados caracterizam o canal transdutor número 1, ou seja, são dados referentes ao processamento associado com o canal e dados típicos do sensor de temperatura LM35. O endereço de função enviado é o código "10100000_2" ou "160_{10}", que corresponde à leitura de dados da estrutura Canal-TEDS. Observa-se que o canal endereçado é o número 1 e que os dados correspondentes são enviados através da TII, por meio do sinal DOUT, em diversos pacotes de 8 *bits*. A transferência de dados acaba ao transferir a estrutura completa. As estruturas Canal-TEDS1 e Canal-TEDS2 podem ser analisadas nas Tabelas 8.3 e 8.4.

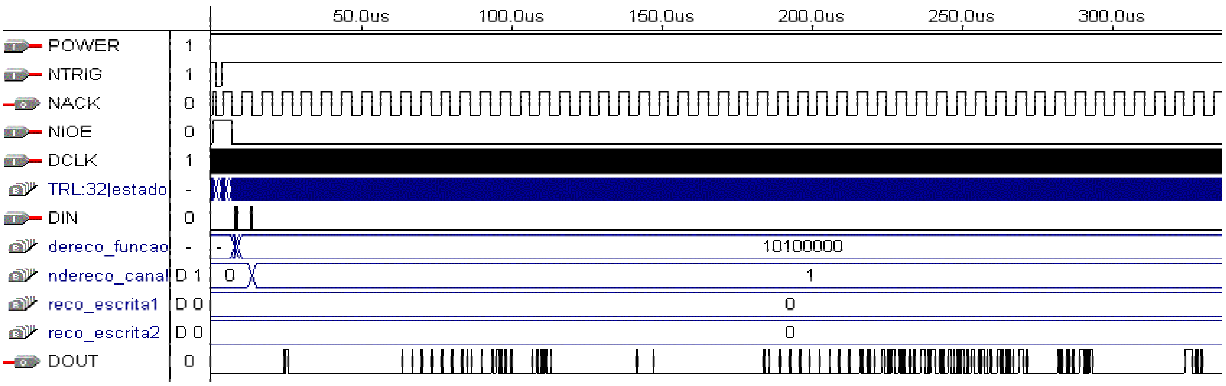


Figura 8.10. Leitura dos dados contidos na estrutura Canal-TEDS.

Tabela 8.3. Estrutura Canal-TEDS1.

Campo	Descrição	Tipo	Nº de Bytes	Conteúdo	Codificação
1	Channel TEDS Length	U32L	4	92	00000000000000000000000001011100
2	Calibration Key	U8E	1	0 (NONE)	00000000
3	Channel Industry Calibration TEDS Extension Key	U8E	1	0	00000000
4	Channel Industry Nonvolatile Data Fields Extension Key	U8E	1	0	00000000
5	Channel Industry TEDS Extension Key	U8E	1	0	00000000
6	Channel End-User's Application-Specific TEDS Key	U8E	1	0	00000000
7	Channel Writable TEDS Length	U32C	4	0	00000000000000000000000000000000
Transducer related data sub-block					
8	Channel Type Key	U8E	1	0 (SENSOR)	00000000
9	Physical Units	UNITS	10	Kelvin	$0,128_2,128_2,128_2,128_2,128_2,128_2,130_2,128_2,128_2$
10	Lower Range limit	F32	4	-273	11000011100010001000000000000000
11	Upper Range Limit	F32	4	333	01000011101001101000000000000000
12	Worst-Case Uncertainty	F32	4	-	-
13	Self-Test Key	U8E	1	0 (NONE)	00000000
Data converter related data sub-block					
14	Channel Data Model	U8E	1	0 (N-Byte)	00000000
15	Channel Data Model Length	U8C	1	1	00000001
16	Channel Model Significant Bits	U16C	2	8	0000000000001000
17	Channel Data Repetitions	U16C	2	0	0000000000000000
18	Series Origin	F32	4	0	00000000000000000000000000000000
19	Series Increment	F32	4	0	00000000000000000000000000000000
20	Series Unit	UNITS	10	0	$4,128_2,128_2,128_2,128_2,128_2,128_2,128_2,128_2,128_2$
Timing related data sub-block					
21	Channel Update Time (t_u)	F32	4	3.00 E –7(seg)	00110100101000010000111110101111
22	Channel Write Setup Time (t_{ws})	F32	4	6.00 E -7	00110101001000010000111110101111
23	Channel Read Setup Time (t_{rs})	F32	4	3.60 E -6	00110110011100011001011110000111
24	Channel Sampling Period (t_{sp})	F32	4	18.65 E -6	00110111100111000111001010010001
25	Channel Warm-Up Time	F32	4	1	00111111100000000000000000000000
26	Channel Aggregated Hold-Off-Time (t_{ch})	F32	4	3.00 E -7	00110100101000010000111110101111
27	Timing Correction	F32	4	0	00000000000000000000000000000000
28	Trigger Accuracy	F32	4	0	00000000000000000000000000000000
Event sequence options field					
29	Event Sequence Options	U8E	1	0	00000000
Data Integrity data sub-block					
30	Checksum for Channel TEDS	U16C	2	65441	1111111110100001

Tabela 8.4. Estrutura Canal-TEDS2.

Campo	Descrição	Tipo	Nº de Bytes	Conteúdo	Codificação
1	Channel TEDS Length	U32L	4	92	00000000000000000000000001011100
2	Calibration Key	U8E	1	0 (NONE)	00000000
3	Channel Industry Calibration TEDS Extension Key	U8E	1	0	00000000
4	Channel Industry Nonvolatile Data Fields Extension Key	U8E	1	0	00000000
5	Channel Industry TEDS Extension Key	U8E	1	0	00000000
6	Channel End-User's Application-Specific TEDS Key	U8E	1	0	00000000
7	Channel Writable TEDS Length	U32C	4	0	00000000000000000000000000000000
Transducer related data sub-block					
8	Channel Type Key	U8E	1	1	00000001
9	Physical Units	UNITS	10	Volts	$0,128_2,128_2,132_2,130_2,122_2,126_2,128_2,128_2,\ 128_2$
10	Lower Range limit	F32	4	4	01000000100000000000000000000000
11	Upper Range Limit	F32	4	12	01000001010000000000000000000000
12	Worst-Case Uncertainty	F32	4	0	00000000000000000000000000000000
13	Self-Test Key	U8E	1	0 (NONE)	00000000
Data converter related data sub-block					
14	Channel Data Model	U8E	1	0 (N-Byte)	00000000
15	Channel Data Model Length	U8C	1	1	00000001
16	Channel Model Significant Bits	U16C	2	1	0000000000000001
17	Channel Data Repetitions	U16C	2	0	0000000000000000
18	Series Origin	F32	4	0	00000000000000000000000000000000
19	Series Increment	F32	4	0	00000000000000000000000000000000
20	Series Unit	UNITS	10	0	$4,128_2,128_2,128_2,128_2,128_2,128_2,128_2,128_2,\ 128_2$
Timing related data sub-block					
21	Channel Update Time (t_u)	F32	4	3.00 E -7	00110100101000010000111110101111
22	Channel Write Setup Time (t_{ws})	F32	4	6.00 E -7	00110101001000010000111110101111
23	Channel Read Setup Time (t_{rs})	F32	4	3.60 E -6	00110110011100011001011110000111
24	Channel Sampling Period (t_{sp})	F32	4	18.65 E -6	00110111100111000111001010010001
25	Channel Warm-Up Time	F32	4	2	01000000000000000000000000000000
26	Channel Aggregated Hold-Off-Time (t_{ch})	F32	4	3.00 E -7	00110100101000010000111110101111
27	Timing Correction	F32	4	0	00000000000000000000000000000000
28	Trigger Accuracy	F32	4	0	00000000000000000000000000000000
Event sequence options field					
29	Event Sequence Options	U8E	1	0	00000000
Data Integrity data sub-block					
30	Checksum for Channel TEDS	U16C	2	65441	1111111110100001

8.1.2.5 - Lendo o Bloco Meta-TEDS

Na Figura 8.11a é mostrado o processo de leitura dos dados contidos na estrutura Meta-TEDS, os quais caracterizam o STIM como sendo um conjunto composto por canais transdutores. O endereço de função enviado é o código "10100000_2" ou "160_{10}" aplicado ao canal 0.

A estrutura Meta-TEDS completa, pode ser analisada na Tabela 8.5, salientando-se que tanto esta estrutura quanto as Canal-TEDS foram escritas em um arquivo de entrada textual com a extensão *.mif* (*Memory initialization File*). Ao programar o projeto em um dispositivo PLD, estes dados são gravados em memória.

Na Figura 8.11b é apresentado o resultado experimental, mostrando o sinal NACK vs.o sinal DOUT.

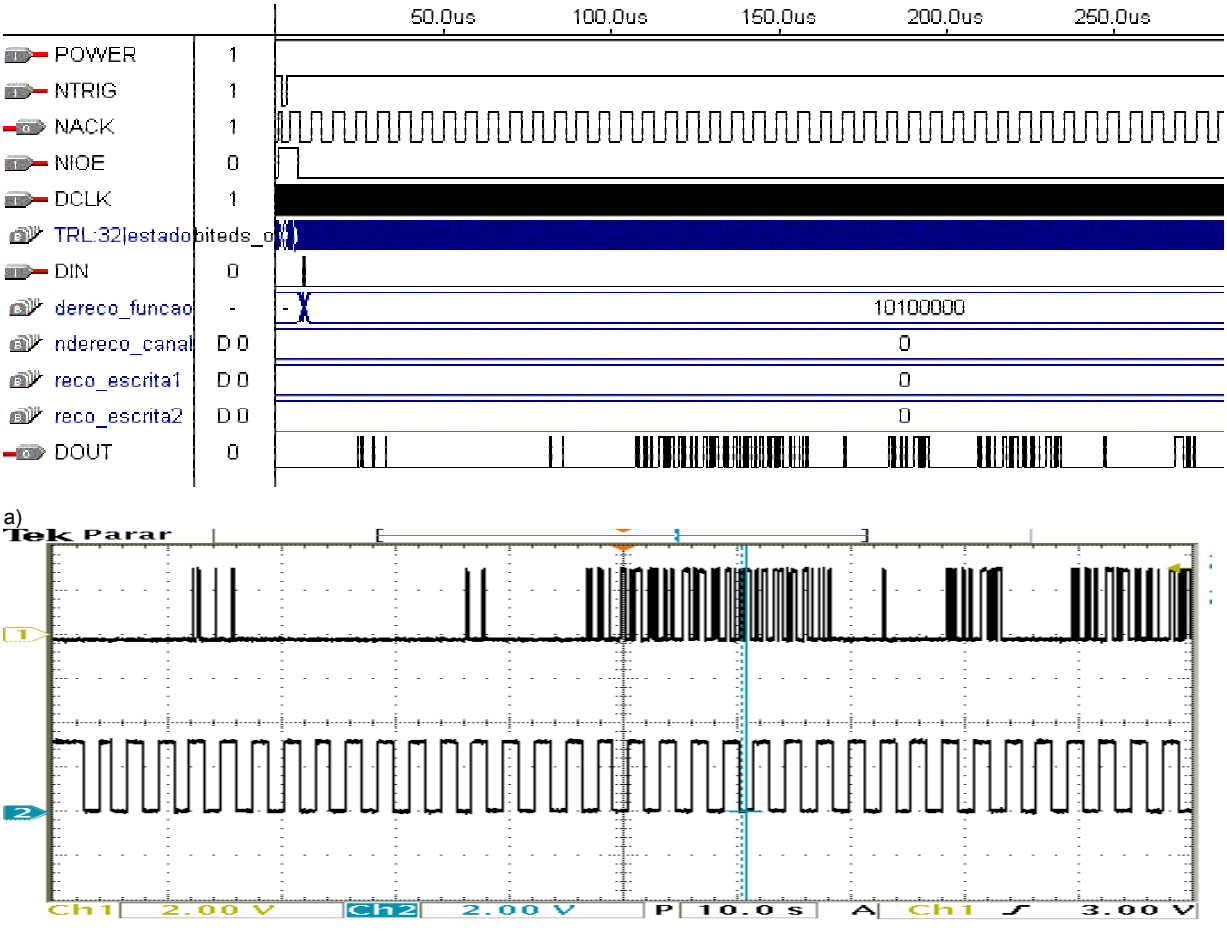


Figura 8.11. Leitura dos dados contidos na estrutura Meta-TEDS: a) simulação; b) resultado experimental.

Tabela 8.5. Estrutura Meta-TEDS.

Campo	Descrição	Tipo	Bytes	Conteúdo	Codificação
TEDS version constant related data sub-block					
1	Meta-TEDS Length	U32L	4	76	00000000000000000000000001001100
2	IEEE 1451 Standards Family Working Group #	U8E	1	2(1451.2)	00000010
3	TEDS Version Number	U8E	1	1(1451.2)	00000001
Identification related data sub-block					
4	Globally Unique Identifier	UUID	10	-	-
Data structure related data sub-block					
5	CHANNEL_ZERO Industry Calibration TEDS Extension Key	U8E	1	0 (NONE)	00000000
6	CHANNEL_ZERO Industry Nonvolatile Data Field Extension Key	U8E	1	0	00000000
7	CHANNEL_ZERO Industry TEDS Extension Key	U8E	1	0	00000000
8	CHANNEL_ZERO End-User's Application-Specific TEDS Key	U8E	1	0	00000000
9	Number of implemented channels	U8C	1	2	00000010
10	Worst-Case Channel Data Model Length	U8C	1	1	00000001
11	Worst-Case Channel Data Repetitions	U16C	2	0	00000000
12	CHANNEL_ZERO writable TEDS Length	U32C	4	0	00000000
Timing related data sub-block					
13	Worst-Case Channel Update Time (t_{wu})	F32	4	3.00 E -7	00110100101000010000111110101111
14	Global Write Setup Time (t_{gws})	F32	4	6.00 E -7	00110101001000010000111110101111
15	Global Read Setup Time (t_{grs})	F32	4	3.60 E -6	00110110011100011001011110000111
16	Worst-case Channel Sampling Period (t_{wsp})	F32	4	18.65 E -6	00110111100111000111001010010001
17	Worst-case Channel Warm-Up Time	F32	4	2	01000000000000000000000000000000
18	Command Response Time	F32	4	20.00 E-6	00110111101001111100010110101100
19	STIM Handshake Timing (t_{hs})	F32	4	6.00 E -7	00110101001000010000111110101111
20	End-of-Frame Detection Latency (t_{lat})	F32	4	0	00000000000000000000000000000000
21	TEDS Hold-Off Time (t_{th})	F32	4	3.00 E -7	00110100101000010000111110101111
22	Operational Hold-Off Time (t_{oh})	F32	4	3.00 E -7	00110100101000010000111110101111
23	Maximum Data Rate	U32C	4	3 (Mbps)	00000000000000000000000000000011
Channel grouping related data sub-block					
24	Channel Groupings Data Sub-Block Length	U16L	2	0	0000000000000000
25	Number of Channel Groupings = G	U8C	1	0	00000000
Fields 26-28 are repeated G times, once for each group					
26	Group Type	U8E	1	0	00000000
27	Number of Group Members = N	U8C	1	0	00000000
28	Member Channel Numbers List = M(N)	Array of U8E	N	0	00000000
Data integrity data sub-block					
29	Checksum for Meta-TEDS	U16C	2	65457	1111111110110001

8.1.2.6 - Acionando o Atuador do Canal Transdutor Número 2

Para ligar o ventilador associado ao canal transdutor número 2, utilizou-se o endereço de função cujo valor é 16 que, de acordo com o padrão 1451, corresponde a operações de escrita e pode ser utilizado para aplicações industriais com critério aberto. O processo para ligar e desligar o ventilador é apresentado no exemplo de simulação da Figura 8.12. Através de DIN deve ser enviado o endereço de função "16_{10}" endereçado para o canal 2, o que fará com que a saída digital para ligar o ventilador assuma o valor lógico "1". Para desligar o ventilador é suficiente aplicar um comando de *reset* sobre o canal 2, ou seja, enviar através de DIN, o endereço de função "1_{10}" (escrever comando de controle), o endereço de canal "2_{10}" e o endereço de escrita "1_{10}", representado por meio de 16 *bits* (comando de *reset*).

O atuador pode ser ligado ou desligado a qualquer momento, ou de acordo com um determinado algoritmo de controle do tipo *on-off* implementado no STIM. Na Figura 8.12 são apresentadas três instâncias seqüenciais diferentes de transferência de dados: A) acionamento do ventilador, B) leitura de dados do canal 1 e C) *reset* do canal 2 para desligar o ventilador.

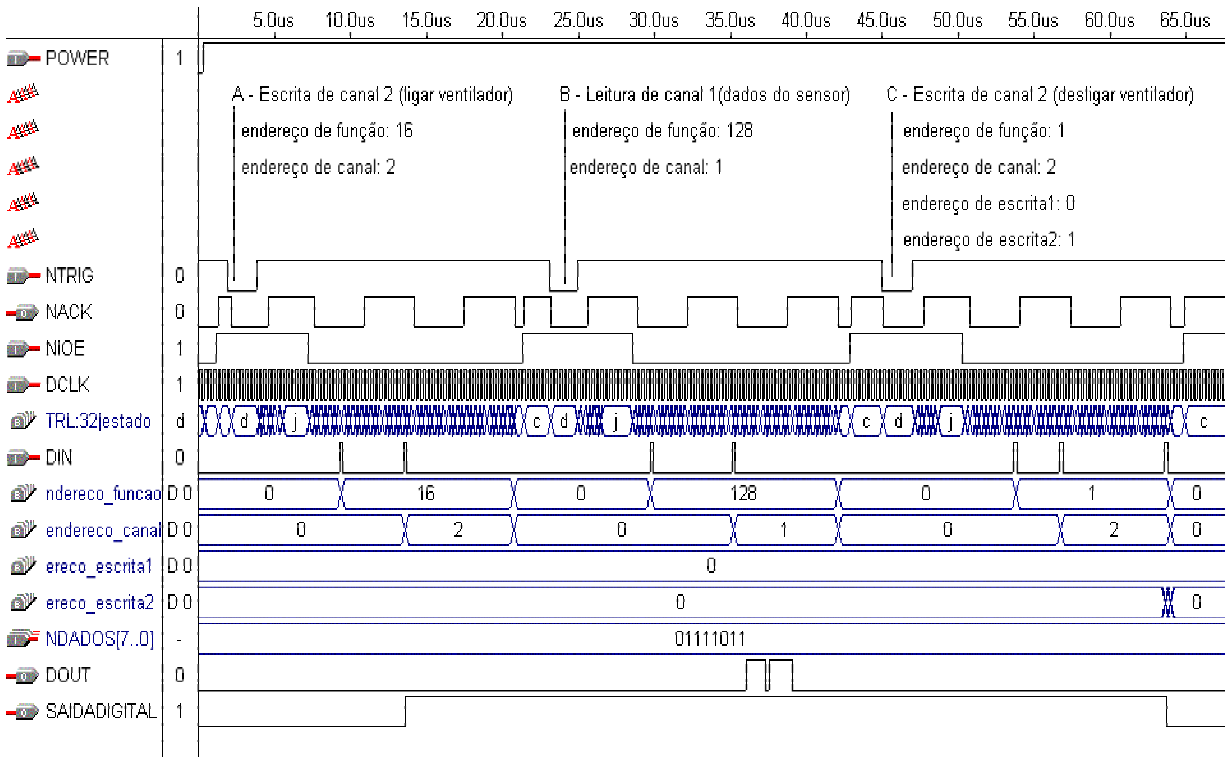


Figura 8.12. Ligando e desligando o atuador do canal transdutor 2.

8.1.2.7- Escrevendo as Máscaras de Interrupção

As máscaras dos registradores de interrupção são escritas através da seguinte seqüência: a) endereço de função "5_{10}" (escrever máscaras de interrupção), b) endereço de canal e c) endereço de escrita, representado por meio de 16 *bits*, através do qual é enviado o valor de escrita correspondente. No exemplo da Figura 8.13, mostra-se a escrita das máscaras de interrupção do canal transdutor 1, com o valor "1111111100000000_2". Isto faz com que seja solicitada uma interrupção através do sinal NINT, pois o primeiro registrador de máscaras de interrupção passa a ter todos os bits em "1".

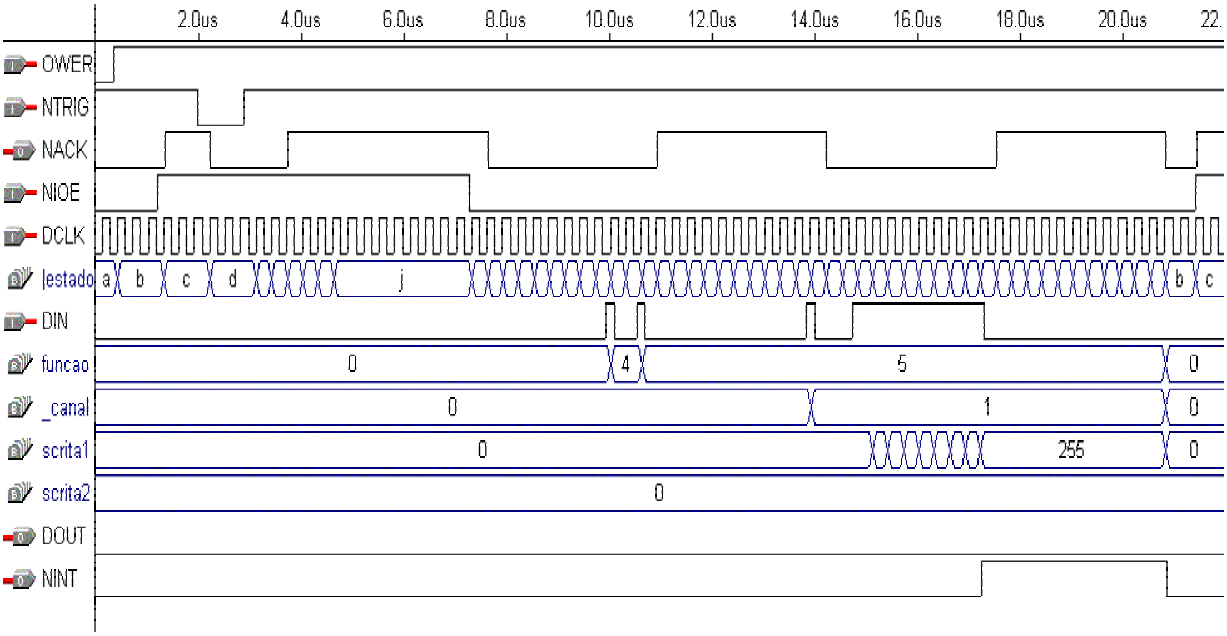
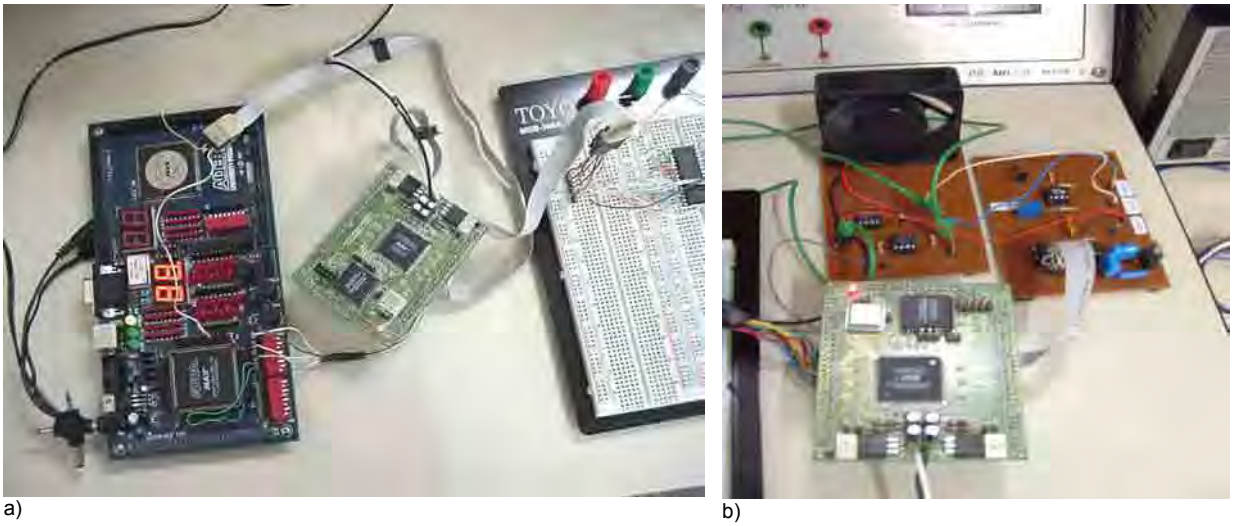


Figura 8.13. Escrita das máscaras de interrupção.

8.1.2.8 - Montagem Realizada em Laboratório

Na Figura 8.14a é mostrado um dos primeiros testes feitos com o STIM implementado com uma FPGA EPIK50TC144-3 e sendo gerenciado por uma FPGA EPF10K20RC240-4.

Na Figura 8.14b é mostrado o STIM contendo os dois canais transdutores com seus respectivos circuitos de conversão e condicionamento de sinais.



a) b)

Figura 8.14. STIM implementado no laboratório.

8.2 - Implementação do NCAP

8.2.1 - Gerenciador de Protocolo com Entrada e Saída de Oito *Bits*

Na Figura 8.15 é mostrada a multiplexação dos dados fornecidos por meio da porta paralela. Observa-se que, quando a entrada de controle assume o valor binário "01", o dado enviado através da porta paralela é colocado no barramento de função. Este evento também fará com que o Gerenciador de Protocolo produza um disparo via sinal NTRIG da TII, a fim de estabelecer o protocolo IEEE1451.2 com o STIM, fato que pode ser analisado na Figura 8.16.

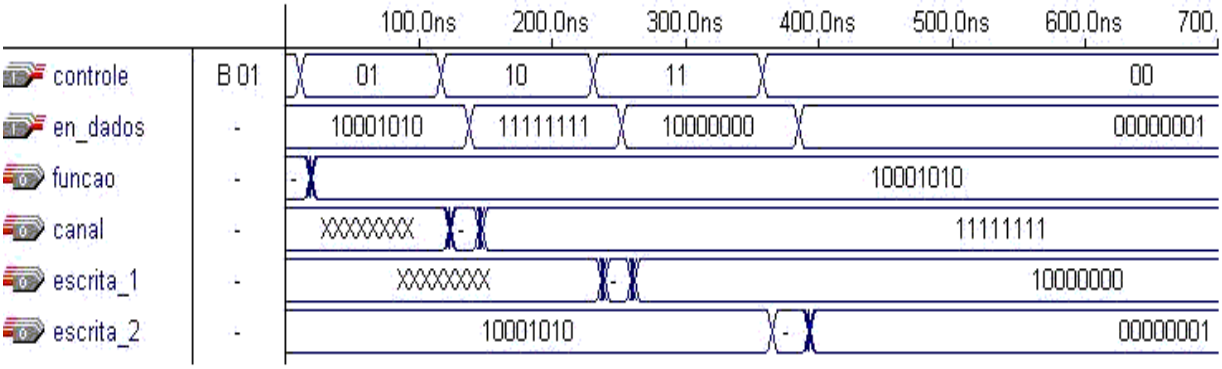


Figura 8.15. Simulação da multiplexação de dados da porta paralela.

Pode-se observar que os códigos enviados através da porta paralela, são colocados nos barramentos de função, canal e escrita de acordo com a entrada de controle correspondente e enviados de forma sincronizada, para o STIM, por meio do sinal DIN, que no caso do Gerenciador de Protocolo, constitui uma saída.

Por seu turno, o STIM envia os valores, através de DOUT e o Gerenciador os coloca nos barramentos SAIDAPARALELA1[7..0] e SAIDAPARALELA2[7..0] respectivamente para processamento interno no NCAP.

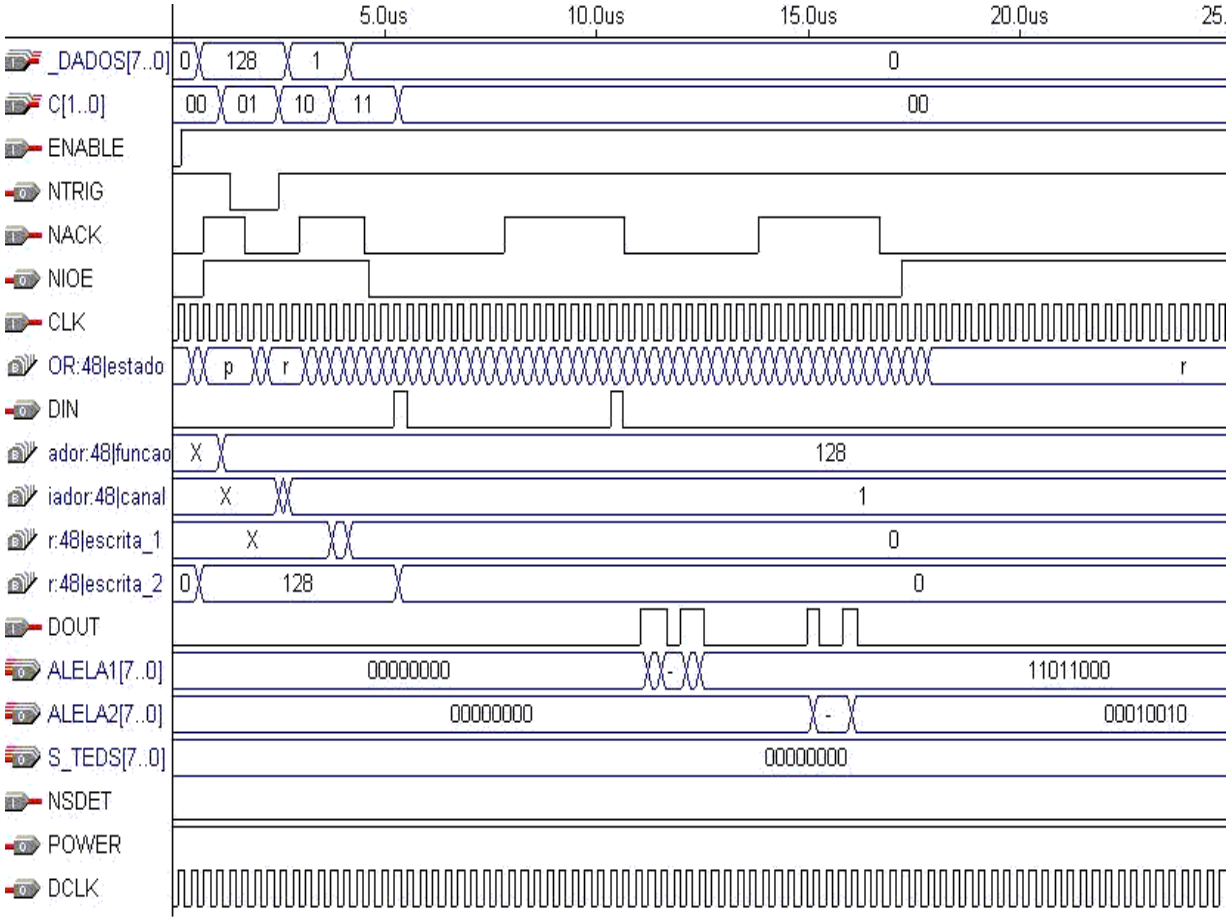


Figura 8.16. Protocolo IEEE 1451.2 do lado NCAP.

O Gerenciador de Protocolo foi implementado com uma FPGA EPF10K20RC240-4 e os recursos utilizados são indicados na Tabela 8.6.

Tabela 8.6. Gerenciador com entrada e saída de 8 *bits*; recursos utilizados.

	Elementos lógicos	I/O
Recursos disponíveis	1152	189
Recursos utilizados	490 (42%)	32 (16,9%)

8.2.2 - Gerenciador de Protocolo com Entrada de Oito e Saída de Cinco *Bits*

Na Figura 8.17 é mostrada a simulação correspondente ao Gerenciador de Protocolo com entrada de 8 *bits* e saída de 5 *bits*. Note-se que quando o endereço 127 aparece na entrada de dados, produz-se um sinal de disparo na linha NTRIG, iniciando-se o protocolo 1451.2. Note-se também como o dado fornecido pelo STIM é divido em duas parcelas de 5 *bits*, com o primeiro *bit* invertido.

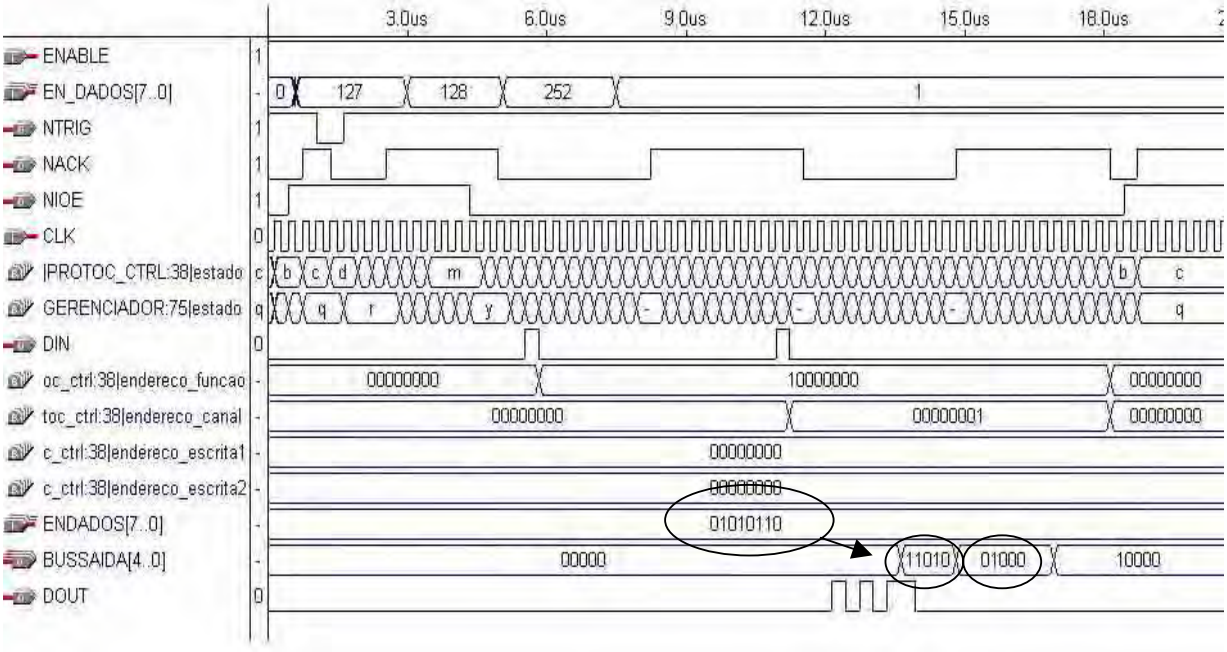


Figura 8.17. Protocolo IEEE 1451.2 do lado NCAP, usando o gerenciador com entrada de 8 *bits* e saída de 5 *bits*.

Esta versão do Gerenciador de Protocolo foi implementada com uma FPGA EPF10K20RC240-4 e os recursos utilizados são indicados na Tabela 8.7.

Tabela 8.7. Gerenciador com entrada de 8 *bits* e saída de 5 *bits*; recursos utilizados.

	Elementos lógicos	I/O
Recursos disponíveis	1152	189
Recursos utilizados	667 (57%)	27 (14,3%)

8.2.3 – *Software* do NCAP Baseado em Java RMI

Na Figura 8.18 são mostradas as etapas necessárias à colocação do servidor no ar. O programa *IEEE1451_NCAPBlock* deve ser compilado através do comando *javac*, logo depois deve ser iniciado o registro RMI. Posteriormente o programa deve ser compilado no ambiente RMI usando o comando *rmic* e, finalmente, executado através do comando *java.* A interface *IEEE1451_BaseTransducerBlock* é compilada automaticamente com o programa *IEEE1451_NCAPBlock.*

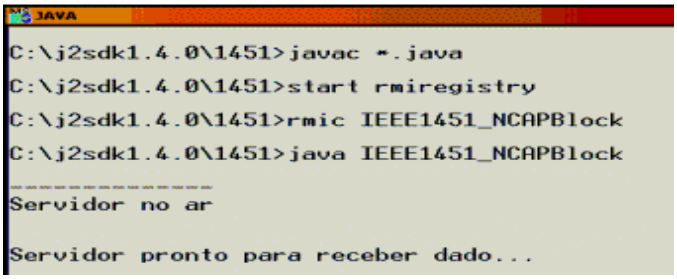


Figura 8.18. Compilação e execução do programa *IEEE1451_NCAPBlock.*

Na Figura 8.19 são mostradas as etapas necessárias à execução do programa cliente *IEEE1451_FunctionBlock.* No instante em que o programa cliente é executado, do lado servidor é informado o endereço IP do cliente que está solicitando serviços, como ilustrado na Figura 8.20.

Figura 8.19. Compilação e execução do programa *IEEE1451 FunctionBlock.*

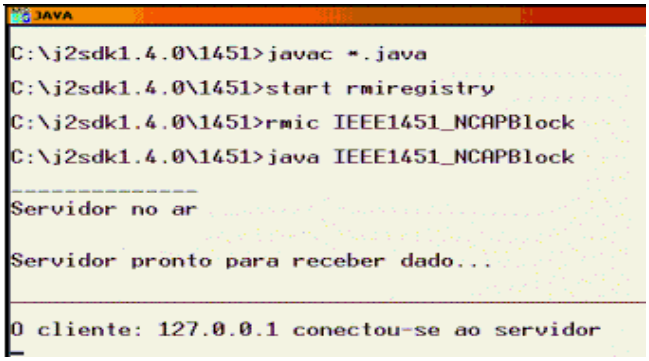


Figura 8.20. Identificação de um cliente no servidor.

Na Figura 8.21 é apresentada uma interface gráfica simples que é exibida ao executar o programa cliente. A interface permite escolher uma opção associada com uma atividade de um canal transdutor e enviá-la para o servidor. Em função do código recebido, o NCAP realiza o endereçamento de acordo com o padrão IEEE 1451. O valor enviado pelo cliente é informado na tela do monitor, como mostrado na Figura 8.22.

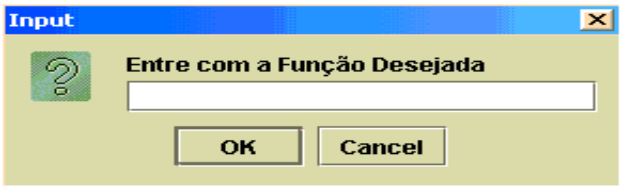


Figura 8.21. Interface gráfica.

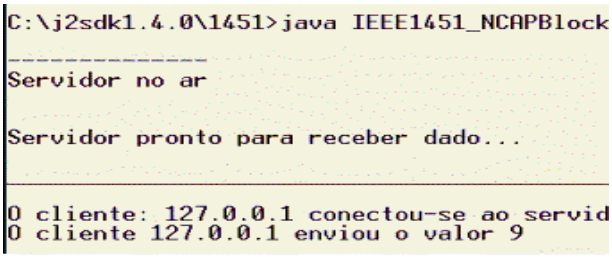
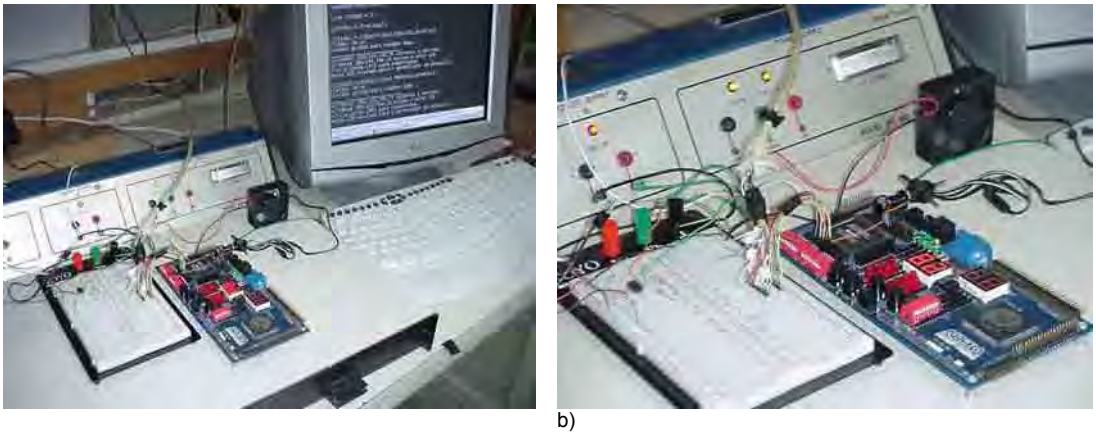


Figura 8.22. O servidor recebe um valor enviado pela aplicação cliente.

Na Figura 8.23 é mostrado o primeiro protótipo empregado para testar o *software* do NCAP. Nesse caso utilizou-se o *software* para ligar e desligar um ventilador de forma remota. Conectou-se uma FPGA EPF10K20RC240-4 à porta paralela de um PC para controlar o acionamento do ventilador.



a) b)

Figura 8.23. Montagem realizada para testar o *software* do NCAP.

8.2.4 – *Software* do NCAP Baseado em Java NET

Na Figura 8.24, 8.25 e 8.26 são mostradas diferentes etapas da interação cliente-servidor com base na API NET e o pacote *Parport*. Na Figura 8.24a, o servidor[41] está aguardando uma conexão por parte de um cliente. Na Figura 8.24b um cliente remoto se conecta ao servidor localizado em "diodo.dee.feis.unesp.br" e, na Figura 8.24c, o servidor aceita a conexão do cliente "picapau.dee.feis.unesp.br" e fica pronto para responder às suas solicitações.

Na Figura 8.25a, o cliente envia a função 128 e o canal 1, requisitando a leitura do sensor de temperatura LM35. Na Figura 8.25b, o servidor aceita o pedido e envia o valor correspondente (28,9 ºC) e, na Figura 8.25c, o cliente recebe o valor enviado pelo servidor.

No exemplo da Figura 8.26, o cliente solicita a leitura das TEDS e, posteriormente, do registrador de estados do canal 1, o servidor responde em conseqüência.

Os testes foram feitos em diferentes plataformas e não foi necessária nenhuma re-configuração. Os testes foram realizados da seguinte maneira:

a) servidor *Windows® 98*, cliente *Windows®XP* em rede local;

b) servidor *Windows® XP*, cliente *Linux Kurumin* em rede local;

c) servidor *Windows® 98*, cliente *Windows® 98* via Internet[42].

---

[41] Para rodar o servidor é necessário compilar e executar apenas a classe *NCAPBlock*. Para rodar o aplicativo cliente no modo *localhost* ou no modo remoto, é necessário compilar e executar a classe *Client*.

[42] Embora não tenha sido testado experimentalmente, o servidor pode ser implementado em *Linux,* pois todos os pacotes utilizados existem na versão *Linux*, inclusive o *Parport*.

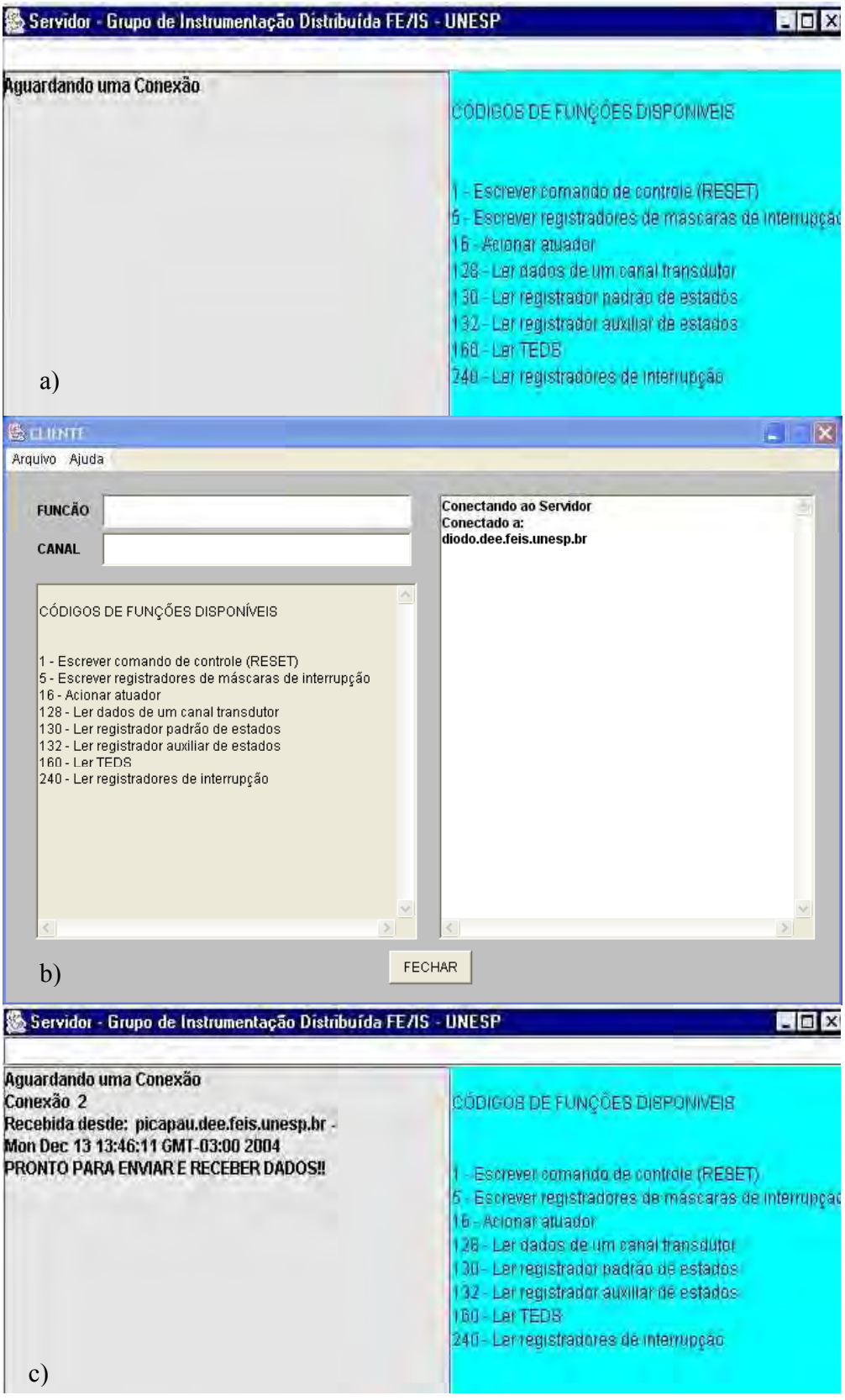


Figura 8.24. Interação cliente-servidor com base na API NET.

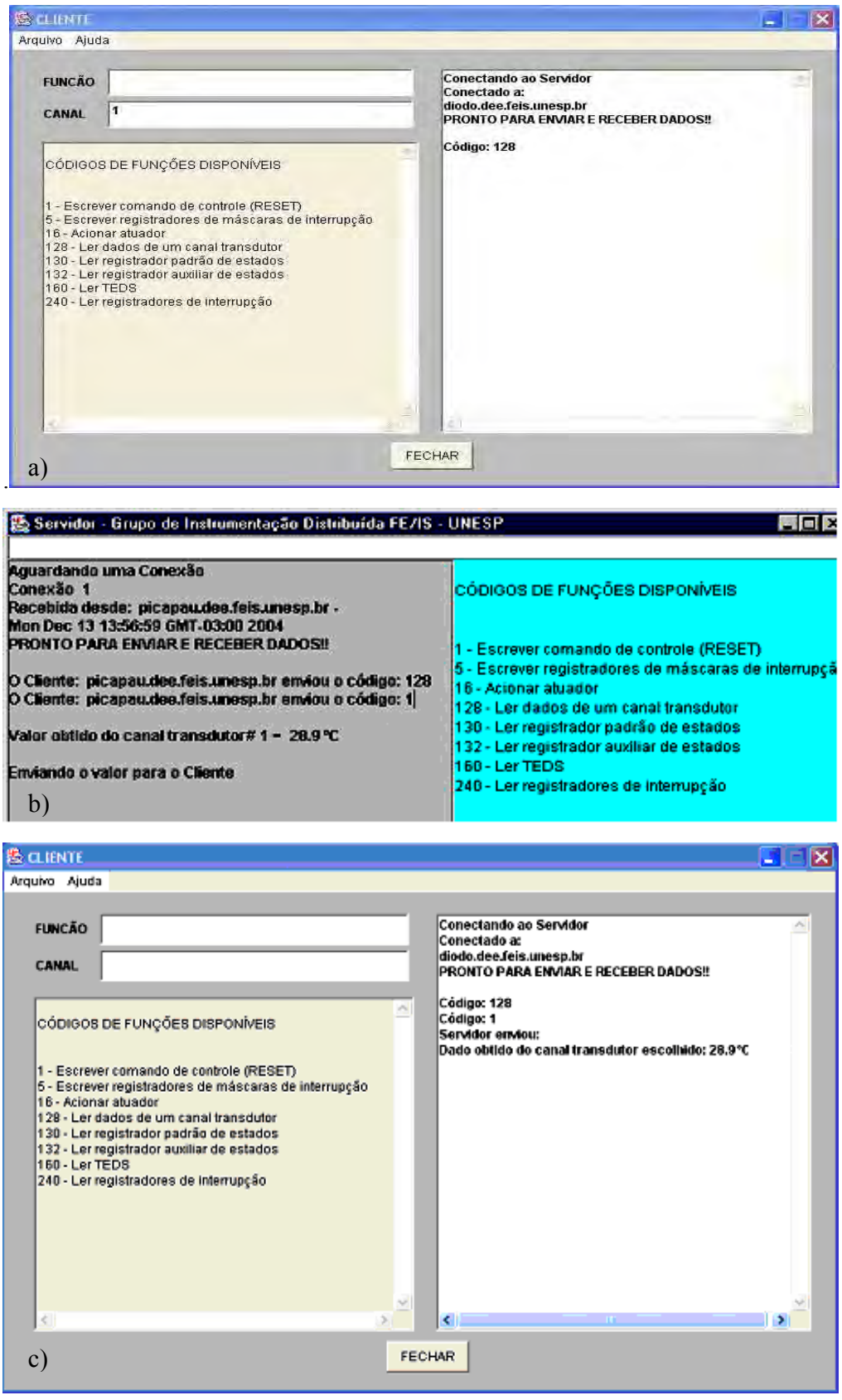


Figura 8.25. Cliente solicitando a leitura do canal transdutor número 1.

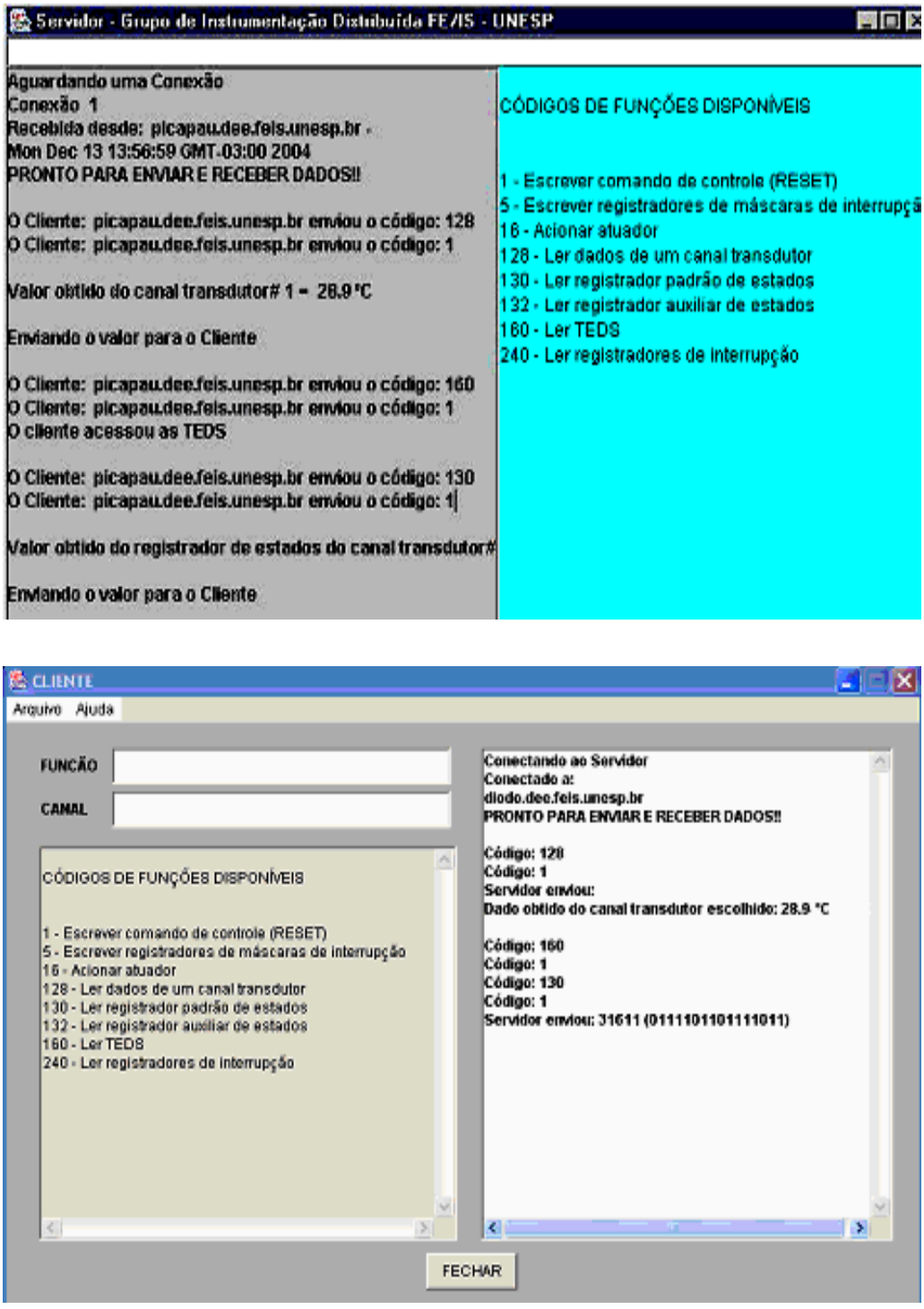
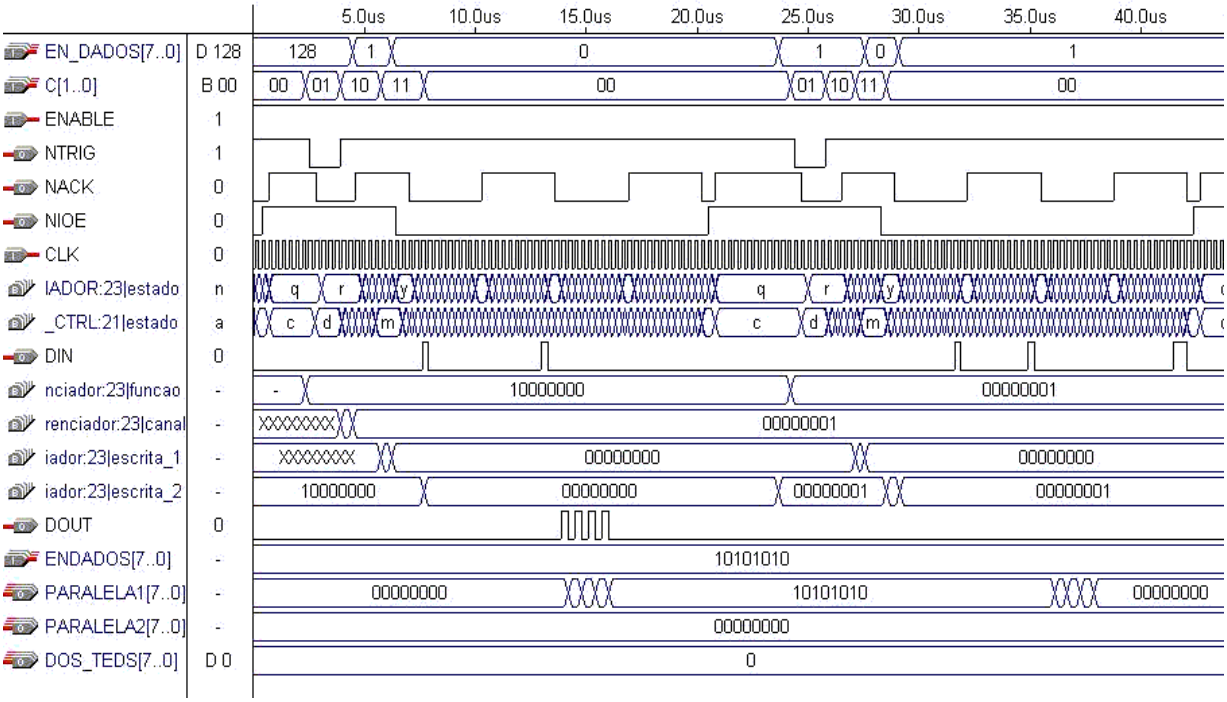


Figura 8.26. Cliente solicitando a leitura das TEDS e do registrador de estados.

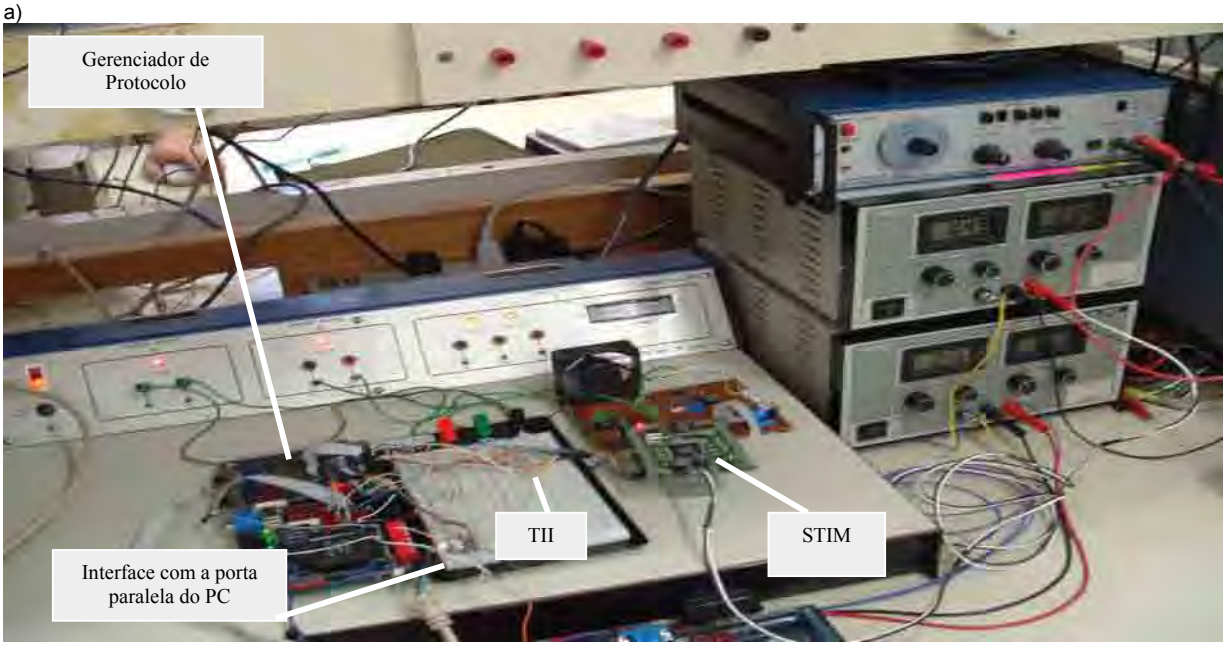
Cabe considerar que, pelo fato das informações TEDS serem muito extensas, quando o cliente solicita a leitura das informações TEDS, elas aparecem na tela do DOS.

8.3 – Nó IEEE 1451 Completo

Na Figura 8.27a é mostrado um exemplo de simulação do Gerenciador de Protocolo conectado ao STIM e, na Figura 8.27b, a montagem realizada no laboratório.



a)



b)

Figura 8.27. Nó IEEE 1451 completo: a) simulação; b) montagem realizada no laboratório.

8.4 – Desempenho do STIM e do Gerenciador de Protocolo

A freqüências máximas teóricas de operação do STIM e do Gerenciador de Protocolo podem ser obtidas através do analisador de tempos dos ambientes de projeto *Max+Plus® II* ou *Quartus® II*.

Logicamente, o STIM possui uma freqüência máxima de operação e o Gerenciador de Protocolo outra, pois este valor depende do projeto e do dispositivo empregado.

Para fins práticos e para esta aplicação em particular interessa a freqüência do sistema STIM-Gerenciador de Protocolo, a qual pode ser calculada instanciando, no ambiente de projeto, ambos os esquemáticos vinculando-os através da TII. Após a compilação do projeto ter sido realizada, o emprego do analisador de tempos e da ferramenta *Clique* para a otimização do roteamento lógico, permitem obter o valor máximo de freqüência de operação do sistema, como mostrado na Figura 8.28.

Figura 8.28. Emprego do analisador de tempos do ambiente *Max+Plus®II* para estimar a freqüência máxima de operação do conjunto STIM-Gerenciador de Protocolo.

Como pode ser analisado na Figura 8.28, a freqüência máxima de operação do sistema STIM-Gerenciador de Protocolo de 8-5 *bits* é 12 MHz. Em termos de taxa de transferência o valor corresponde a 12 Mbps.

8.5 – Comprovação do Modo *Plug and Play*

Com a finalidade de comprovar experimentalmente o modo de operação *plug and play* do STIM foram conectados e retirados três diferentes STIMs, do conector da TII vinculado ao Gerenciador de Protocolo. O teste do modo *plug and play* realizou-se através das alternativas mencionadas a seguir:

a) O STIM apresentado nos Capítulos 7 e 8, contendo dois canais transdutores;

b) Um STIM implementado com uma FPGA, cujas características são as mesmas do FPGA usado no item a); com dois canais transdutores, um canal contendo um sensor de temperatura e um canal contendo um sensor de umidade relativa;

c) Um STIM implementado com um microcontrolador da *Motorola* com um canal transdutor conectado a um potenciômetro[43].

A comprovação do modo *plug and play* foi feita da seguinte forma:

1) O teste foi realizado a partir da configuração mostrada na Figura 8.27, ou seja, um STIM implementado com uma FPGA ACEX-EPIK50TC144-3, com um canal transdutor de temperatura, contendo um sensor de uso geral modelo LM35, e um canal transdutor contendo um ventilador, conectado ao Gerenciador de Protocolo por meio da conexão física da TII.

2) Sem tirar a alimentação do gerenciador de protocolo, procedeu-se a retirar o STIM do conector da TII e conectar outro STIM, que foi chamado de STIM nº2. O STIM nº2 foi implementado com uma FPGA ACEX-EPIK50TC144-3, com um canal transdutor de temperatura, contendo um sensor PRT (*Platinum Resistance thermometer*) de 1000 Ω, em conformidade com a norma IEC 751, com faixa de medidas de –40 ºC a 60 ºC, e um canal transdutor contendo um sensor de umidade relativa modelo HUMICAP® 180, com faixa de medidas de 0 a 100%. Ambos sensores fazem parte do sistema HMP45C da *Campbell Scientific Corp*., cujas especificações principais são apresentadas no Apêndice F junto aos circuitos de condicionamento implementados para esta aplicação em particular. A configuração montada é mostrada na Figura 8.29.

Não foi necessária nenhuma reconfiguração nesta etapa, basta apenas que o sinal NSDET fornecido pelo STIM nº2 esteja em nível lógico ‘0’, para o gerenciador reconhecer o novo STIM.

---

[43] A implementação de um STIM utilizando microcontrolador faz parte do trabalho do aluno Eduardo Marques, mestrando pelo Programa de Pós-Graduação em Engenharia Elétrica da FEIS/UNESP. O trabalho intitula-se: Implementação de um Módulo de Interface para Transdutores Inteligentes - STIM - Empregando Microcontrolador.

Cabe salientar que a auto-identificação do novo STIM no sistema é atingida sem problemas, pois o STIM nº2 contém as informações TEDS correspondentes ao sistema sensor HMP45C.

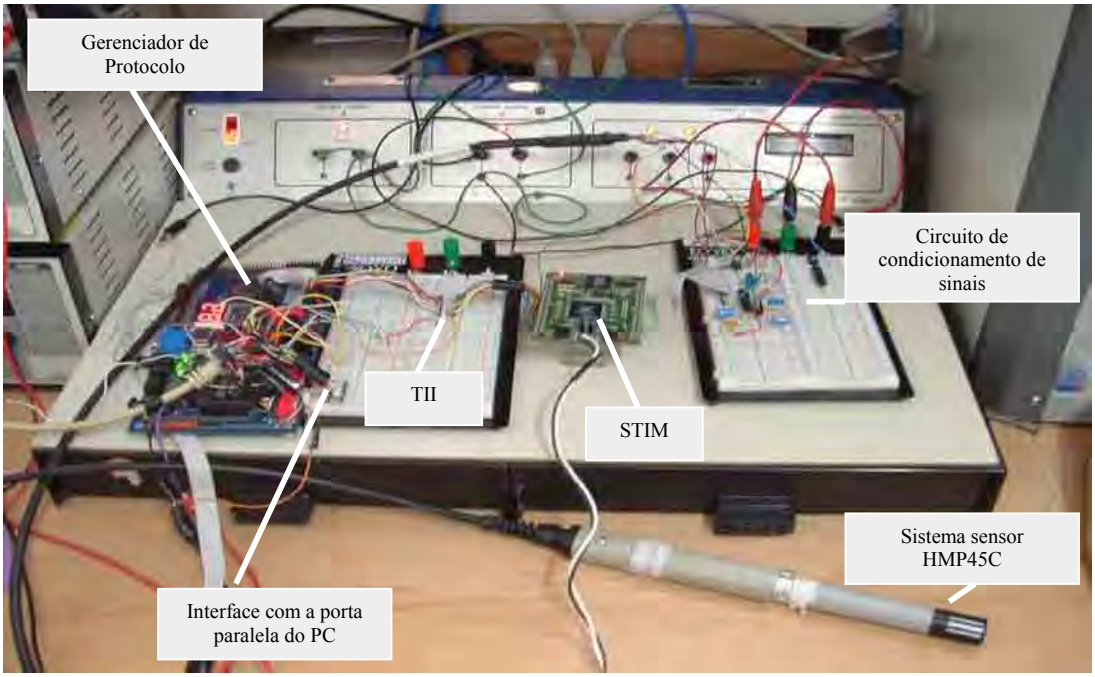


Figura 8.29. Comprovação experimental do modo *plug & play* do STIM.

3) Finalmente, foi retirado o STIM nº2 e conectado um STIM, chamado de STIM nº 3, implementado com um microcontrolador MC68HC908QY4® da Motorola. O STIM nº3 contém uma implementação mínima IEEE 1451.2 com um canal conectado a um potenciômetro, simulando uma entrada analógica variável. Como no caso anterior, o sinal NSDET fornecido pelo STIM nº3 deve permanecer em nível lógico '0', para o gerenciador poder reconhecer o novo módulo.

Neste caso foi necessária uma reconfiguração mínima na temporização do sinal NACK, realizada no programa do STIM nº3, descrito na linguagem *assembly* do microcontrolador.

Na Figura 8.30 é mostrada a conexão do Gerenciador de Protocolo com o microcontrolador. O diagrama do microcontrolador é mostrado na Figura 8.31, apresentando a equivalência usada entre os pinos do dispositivo e os sinais da TII. Desta maneira, a seção 8.5 é finalizada concluindo que o modo de operação *plug and play* do STIM foi comprovado experimentalmente com duas tecnologias diferentes: dispositivos lógicos programáveis e microcontrolador.

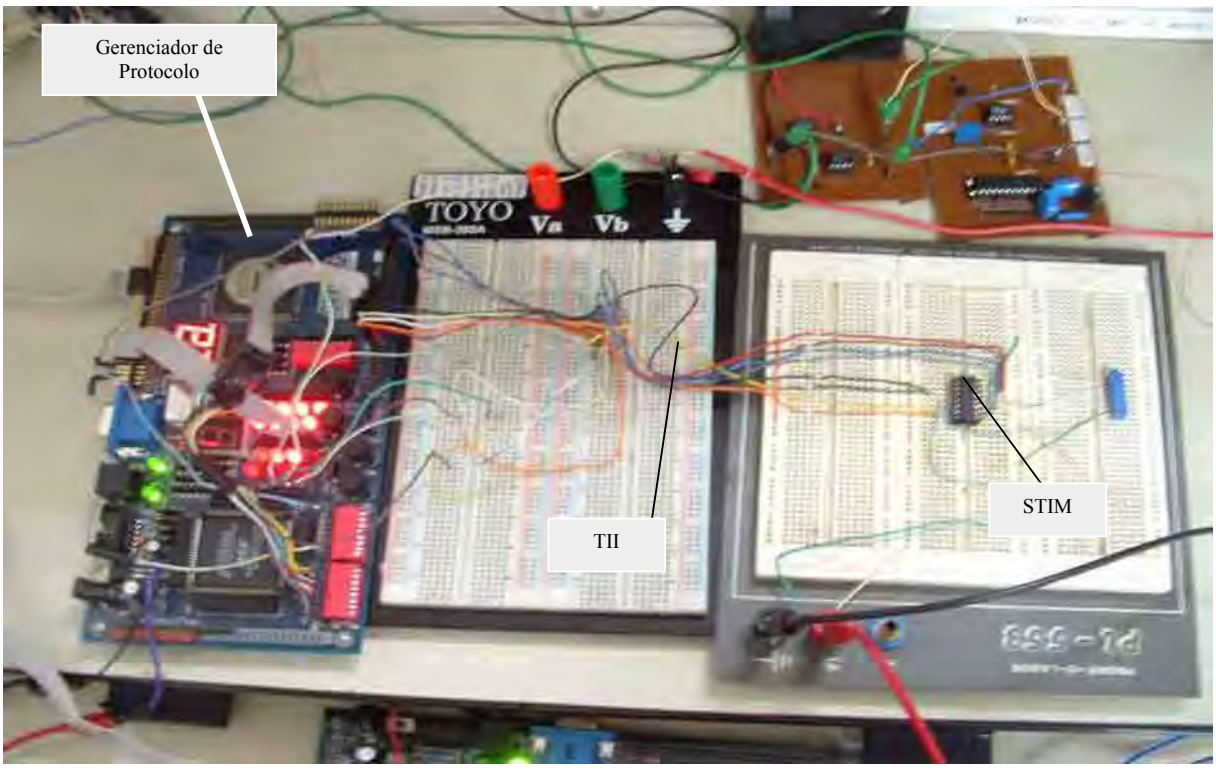


Figura 8.30 . Comprovação experimental do modo *plug & play* de um STIM baseado em microcontrolador.

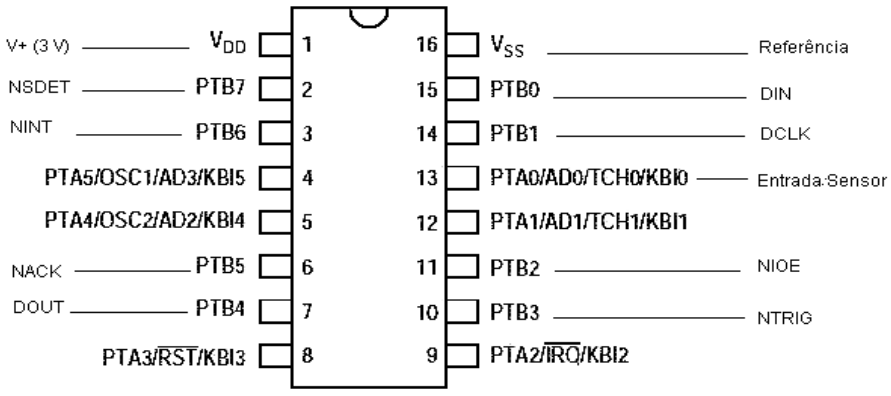


Figura 8.31 . Equivalência entre os pinos do microcontrolador MC68HC908QY4® e os sinais da TII.

Fonte: Motorola. M68HC08 Microcontrollers. Ver 0.1, 12/2002.

8.6 – Comentários Finais sobre o Capítulo 8

Neste capítulo foram apresentados os resultados em relação à metodologia de projeto apresentada no Capítulo 7. Com base nos resultados obtidos pode-se afirmar que as ferramentas utilizadas possuem grande potencial para o desenvolvimento de sistemas na área de instrumentação eletrônica. Impulsionadas pelo crescimento acentuado da Internet, as ferramentas de domínio público são hoje uma realidade, que aplicadas de forma conveniente podem vir a resolver grande parte dos problemas atuais da indústria na área de automação, fundamentalmente no que diz respeito ao custo dos sistemas de instrumentação.

No Capítulo final serão analisados os resultados junto com as conclusões obtidas ao longo do desenvolvimento do presente trabalho e serão propostos alguns trabalhos futuros com base na experiência obtida neste trabalho.

Capítulo 9

Conclusões Gerais

9.1 - Conclusões

No presente trabalho de tese foi apresentado o desenvolvimento de um nó de rede baseado no padrão de interfaceamento IEEE 1451, utilizando ferramentas abertas e padronizadas. O nó pode ser usado em ambientes de instrumentação distribuída sob o modelo de comunicação cliente-servidor com tecnologias de rede local.

Como parte do desenvolvimento do nó de rede foi implementado um STIM de acordo com o padrão de interfaceamento IEEE 1451.2. O módulo foi descrito na linguagem VHDL e sintetizado com dispositivos de lógica programável versáteis e de uso geral, com cerca de 50.000 portas típicas. O projeto foi totalmente criado e simulado nos ambientes de desenvolvimento para projetos digitais *Max+Plus® II* e *Quartus® II* da empresa *Altera*.

Em contraste com as aplicações de microcontroladores, ASICs e soluções comerciais, o emprego de linguagens de descrição de *hardware* e a utilização de dispositivos lógicos programáveis aparecem como uma alternativa muito interessante para construir este tipo de sistema, fundamentalmente pela flexibilidade introduzida no projeto. Este tipo de abordagem possibilita introduzir aspectos relevantes na criação de componentes como reutilização de código e independência tecnológica.

Outro aspecto importante diz respeito à implementação da memória ROM para alocar as TEDS, através de um componente parametrizável que faz parte da biblioteca de megafunções do ambiente. Desta maneira, o componente foi instanciado dentro da hierarquia do STIM. Não houve, portanto, a necessidade de se implementar um dispositivo de memória como periférico.

Para a alternativa apresentada, as simulações e os testes de laboratório mostraram que para uma implementação mínima do padrão IEEE 1451.2, com dois canais transdutores, o projeto pode ser sintetizado em um dispositivo de lógica programável da família FLEX10K com cerca de 20.000 portas típicas. A implementação de mais canais transdutores exige o aumento da

capacidade do dispositivo lógico programável, bem como dos recursos de memória, requisitando a utilização de dispositivos de maior capacidade. Nesse caso, componentes da família ACEX1K com cerca de 50.000 portas típicas podem ser empregados. Para implementar mais canais, o código fonte existente pode ser utilizado com pouco esforço adicional, pois o código é escalável.

Implementou-se um NCAP suportado por um PC convencional. Como parte constituinte do NCAP desenvolveu-se um dispositivo Gerenciador de Protocolo em VHDL, com o intuito de construir um dispositivo simples que possa ser empregado em diversas aplicações e possibilite o modo de operação *plug and play* do STIM. O Gerenciador pode ser conectado a qualquer PC que implemente a funcionalidade do *software* do NCAP desenvolvido neste trabalho. Introduz-se, deste modo, uma grande flexibilidade no sistema, pois a porta paralela não precisa ser modificada e, além disso, o Gerenciador pode ser implementado com um PLD de baixo custo e uso geral.

Através do emprego de um PC para implementar a parte de *hardware* do NCAP podem ser aproveitados recursos já existentes, como *driver* de rede, portas de comunicação e sistema operacional, além de constituir uma ferramenta didática muito flexível.

Empregou-se a plataforma JDK pertencente à tecnologia Java para desenvolver a parte lógica do NCAP, pois é uma ferramenta de domínio público e, além disso, Java é uma linguagem puramente orientada a objetos, que introduz portabilidade, possui relação direta com a Internet e tem perspectivas de crescimento significativo para o futuro.

A utilização da *Ethernet* como rede de controle justifica-se pelo fato de possibilitar a construção de uma estrutura plana de automação, ter baixo custo e ser padronizada através do IEEE 802.3. Embora o problema de determinismo da rede *Ethernet* produzido pelo seu mecanismo de acesso ao meio (CSMA/CD), possa ser um limitante para aplicações industriais em tempo real, as novas especificações da *Ethernet* irão torná-la uma ferramenta muito atrativa para constituir a nova geração de redes para aplicações em sistemas de instrumentação distribuída, com base no protocolo TCP/IP como meio de transporte da informação.

A implementação do padrão IEEE 1451 possibilita aos transdutores desempenhar suas funções em rede de comunicação, proporcionando aos sistemas de monitoramento remoto “inteligência” e integração entre diferentes dispositivos.

Os futuros sistemas de instrumentação e controle distribuído deverão possuir características tais como modularidade, interoperabilidade, serem distribuídos, baseados em padrões, abertos, e seguros. Nesse contexto, a vantagem potencial relacionada com a implementação do padrão IEEE

1451 é a independência que introduz quanto à escolha de transdutores e quanto à escolha da rede, possibilitando aos fabricantes e usuários uma maior flexibilidade e significativa redução de custos por reprojeto de interfaces.

Através da implementação do padrão IEEE 1451 os setores industriais poderão obter inúmeros benefícios, tais como redução de custos de sistemas de instrumentação devido à redução da complexidade, flexibilidade para transmitir informações através de uma rede de comunicação e maior facilidade para ampliar seus sistemas sem efetuar grandes configurações.

As ferramentas utilizadas: VHDL, Java, *Linux*, *Parport*, *Ethernet* e TCP/IP possuem grande potencial para o desenvolvimento de sistemas na área de instrumentação eletrônica, e o que é mais importante, algumas são padronizadas e outras são de domínio público.

Outras ferramentas de domínio público começaram a ser empregadas pelo grupo de pesquisa, dando continuidade a este trabalho. O desenvolvimento de um *website* dinâmico e interativo para monitorar aplicações IEEE 1451, via Internet, é um complemento do trabalho de tese e emprega ferramentas tais como a linguagem PHP (*Personal Home Page-Hypertext Preprocessor*) e MySQL (*My Structured Query Language*) [106]. PHP é uma linguagem de criação de *scripts* do lado do servidor que foi projetada especificamente para o desenvolvimento de aplicações que requerem desempenho e segurança sem custo operacional. Por seu turno, o MySQL é um sistema de Gerenciamento de Banco de Dados Relacional (RDBMS) poderoso e muito rápido.

Certamente, as ferramentas empregadas no desenvolvimento da tese possuem limitações, mas, em contrapartida, possuem vantagens significativas, fundamentalmente no que diz respeito ao custo e à flexibilidade.

O emprego da linguagem Java para construir a parte lógica do NCAP constitui um ponto interessante, pois existe uma carência notável de aplicações da linguagem Java em baixo nível, fato que é possibilitado pelo uso da API COMM. Embora a API de comunicações se apresente hoje como uma tecnologia com falta de amadurecimento, possui um grande potencial para aplicações industriais e, o constante desenvolvimento e atualização da tecnologia Java fará, sem dúvida, que se torne uma ferramenta útil e necessária.

Aplicações com base em Java RMI podem apresentar algumas limitações para aplicações industriais, fundamentalmente pela falta de determinismo. Entretanto, como mostrado neste trabalho de tese, para aplicações baseadas em acionamentos remotos via rede local, que não necessitem satisfazer tempos de resposta rigorosos, Java RMI é uma excelente ferramenta,

principalmente se utilizada junto ao pacote *Parport*. O emprego da API de comunicações é relativamente fácil para operações de escrita, no entanto, para operações de leitura com a porta paralela não tem-se conseguido um comportamento satisfatório, fato este que se atribui a um possível conflito da API com o modo bidirecional da porta paralela. Em contraste, o uso do pacote *Parport* resolve o problema e, ainda, melhora a eficiência do sistema, pois, em seu nível mais baixo, utiliza a linguagem C que possui melhor desempenho do que Java em termos de velocidade.

O *software* com base em Java NET mostra-se muito eficiente tanto em rede local quanto via Internet e usado junto ao pacote *Parport* pode resolver um amplo conjunto de problemas na área de instrumentação baseada em aplicações remotas, não necessariamente em conformidade com o padrão IEEE 1451.

A tecnologia *Ethernet* junto ao conjunto de protocolos TCP/IP pode apresentar problemas de determinismo, porém esses problemas podem ser eliminados ou minimizados diminuindo o número de nós conectados à rede e utilizando a *Ethernet* comutada. A tecnologia *Ethernet* junto ao conjunto de protocolos TCP/IP possuem três características fundamentais: capacidade de integração, baixo custo e conformidade com um padrão.

No sistema desenvolvido, a taxa de transferência de dados sobre a TII pode chegar a um valor máximo de 12 Mbps, entretanto, a aplicação fica limitada à velocidade da porta paralela. No entanto, o sistema desenvolvido tem um amplo campo de aplicações como por exemplo:

a) Aplicações industriais que não requisitem tempos de resposta extremamente rigorosos;

b) Sensoriamento de parâmetros em máquinas elétricas;

c) Controle de temperatura, nível, pressão, etc;

d)Aplicações em automação residencial: monitoramento, portão eletrônico, sistema de iluminação, alarmes, etc.

9.2 - Contribuições

A contribuição mais importante do trabalho apresentado está relacionada com a utilização de ferramentas padronizadas e de domínio aberto no desenvolvimento do nó IEEE 1451 completo.

Esta pode ser considerada a contribuição global do trabalho, entretanto há uma série de contribuições específicas tais como:

a) A implementação do STIM com PLDs versáteis e de uso geral, possibilitando a implementação de sistemas IEEE 1451.2 de baixo custo;
b) A forma de implementação das TEDS, usando uma memória parametrizável descrita em VHDL e o mecanismo de acionamento das memórias, o qual é um sistema simples baseado em contadores;
c) O desenvolvimento e implementação de um dispositivo Gerenciador de Protocolo que funciona em conjunto com a porta paralela de um PC convencional, cujo objetivo é gerenciar o protocolo IEEE 1451.2 e introdzir portabilidade;
d) A implementação do Gerenciador de Protocolo com uma FPGA de uso geral de 20.000 portas típicas;
e) O desenvolvimento do *software* do NCAP integralmente na linguagem Java.

9.3 – Sugestões de Trabalhos Futuros

Como sugestões de trabalhos futuros para o grupo de pesquisa propõem-se os seguintes:
a) Desenvolvimento de sistemas de instrumentação com base nas diretrizes IEEE 1451.3 e 1451.4;

b) Exploração das especificações Java em tempo real (rtj) e suas aplicações pontenciais;

c) Padrão de comunicação SOAP (*Simple Object Access Protocol*): exploração das possíveis aplicações no projeto, por exemplo, para comunicar um servidor *Phyton* com um cliente Java, através da rede.

d) Análise de desempenho da arquitetura *Ethernet*-TCP/IP usando aplicações IEEE 1451;

e) Emprego de tecnologia embarcada, por exemplo, para embutir o *software* do NCAP em um processador Java;

f) Segurança em sistemas de instrumentação distribuída sob o modelo de comunicação cliente-servidor empregando *Ethernet* e Internet

g) Exploração da diretriz IEEE P1451.5 para utilização em redes *wireless* e a potencial aplicação da tecnologia Java para sistemas embarcados, J2ME;

h) Análise de desempenho de STIMs implementados com PLDs vs. STIMs implementados com microcontroladores e placas comerciais;

i) Emprego da linguagem *Phyton* para desenvolver o *software* do NCAP;

REFERÊNCIAS

[1] LEE, K. Sensor networking and interface standardization. In: INSTRUMENTATION AND MEASUREMENT TECHNOLOGY CONFERENCE - IMTC, 18, 2001, Budapest. **Proceedings of the...**, Budapest: IEEE, 2001, v.1, p. 147 – 152.

[2] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, Instrumentation and Measurement Society. **IEEE Standard for a Smart Transducer Interface for Sensor and Actuators - Network Capable Application Processor (NCAP) Information Model, (Std. 1451.1)**. Standards Board. NY: IEEE, 1999. 341 p.

[3] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, Instrumentation and Measurement Society. **IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Transducer to Microprocessor Communication Protocols and Transducer Data Sheet (TEDS) Formats, (Std. 1451.2)**. Standards Board. NY: IEEE, 1997. 114 p.

[4] LEE, K.B.; SCHNEEMAN, R.D. Distributed measurement and control based on the IEEE 1451 smart transducer interface standards. **IEEE Trans. Instrumentation & Measurement**, IEEE, v.49, no. 3, p.621-627, June 2000.

[5] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. IEEE 1451 website. Disponível em: <**http://ieee1451.nist.gov**>. Acesso em: 2002/2003/2004.

[6] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, Instrumentation and Measurement Society. **IEEE Standard for a Smart Transducer Interface for Sensor and Actuators - Digital Communication and Transducer Electronic Data Sheet (TEDS) Formats for Distributed Multidrop Systems, (Std. 1451.3)**. Standards Board. NY: IEEE, 2003. 185p.

[7] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. IEEE Standards website. Disponível em: <**http://standards.ieee.org/announcements/pr_14514.html**>. Acesso em: 2004

[8] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS P1451.5 PROJECT. IEEE P1451.5 website. Draft Standard for a Smart Transducer Interface for Sensor and Actuators - Wireless Communication Protocols and Transducer Electronic Data Sheets. Disponível em: <**http://grouper.ieee.org/groups/1451/5/** >. Acesso em 2003

[9] JOHNSON, R. N. Proposed IEEE standard P1451.0: defining the core features of smart sensors to facilitate broader adoption. Papers in PDF format, Telemonitor Inc., June 2003. Disponível em: <**www.telemonitor.com/doc/dot2.pdf**>. Acesso em: 2003.

[10] SENSORSPORTAL. Sensorsportal website. Standardization. Disponível em: <**http://www.sensorsportal.com/HTML/standard.htm**>. Acesso em: 2004.

[11] ROSSI, S. R.; BATISTA, E. A.; CARVALHO, A. A.; SILVA, A. C. R. IEEE 1451 node development for connecting transducers to networks. In: CONFERÊNCIA INTERNACIONAL DE APLICAÇÕES INDUSTRIAIS - INDUSCON, 6, 2004. **Proceedings of the…**, Joinville, p.361-366.

[12] ROSSI, S. R.; BATISTA, E. A.; CARVALHO, A. A.; SILVA, A. C. R. Utilização da tecnologia de lógica programável na implementação de um módulo de interface para transdutores inteligentes (STIM) IEEE 1451.2. In: CONGRESSO BRASILEIRO DE AUTOMÁTICA - CBA, 15, 2004. **Proceedings of the...**, Gramado, Código: 630.

[13] BATISTA, E. A.; ROSSI, S. R.; SILVA, A. C. R.; CARVALHO, A. A.; KITANO, C. Implementation of a Java Language Program for Developing the Entire Network Capable Application Processor (NCAP) software. In: CONFERÊNCIA INTERNACIONAL DE APLICAÇÕES INDUSTRIAIS - INDUSCON, 6, 2004. **Proceedings of the...**, Joinville, p.382-387.

[14] LEE, K.B.; SCHNEEMAN, R.D. Implementing a standard-based distributed measurement and control application on the Internet. Papers in PDF format, June 1999. Disponível em: <**ieee1451.nist.gov/framework.pdf**>. Acesso em: 2002.

[15] FELSER, M.; SAUTER, T. The fieldbus war: history or short break between battles. In: INTERNATIONAL FACTORY COMMUNICATION SYSTEMS WORKSHOP, 4, 2002. IEEE, 2002, p. 73 - 80.

[16] LEE, K.B.; SCHNEEMAN, R.D. Internet-based distributed measurement and control. **IEEE Applications Instrumentation & Measurement Magazine**, IEEE, v.2, p.23-27, June. 1999.

[17] LEE, K. IEEE 1451: A standard in support of smart transducer networking. In: INSTRUMENTATION AND MEASUREMENT TECHNOLOGY CONFERENCE-IMTC, 17, 2000, Baltimore. **Proceedings of the...**, Baltimore: IEEE, 2000, v.2, p. 525 – 528.

[18] SENSORSPORTAL. Sensorsportal website. Marketplace. Disponível em: <**www.sensorsportal.com**>. Acesso em: 2004.

[19] EDISON, J.; WOODS, S.P. A research prototype of a networked smart sensor system. Papers in PDF format, Measurement Systems Department-Instruments and Photonics Laboratory (HP), August 1995. Disponível em: <**www.hpl.hp.com/techreports/95/HPL-95-91.pdf**>. Acesso em: 2002.

[20] LEE, K.B.; SCHNEEMAN, R.D. Standardized approach for transducer interfacing: implementing IEEE-P1451 Smart Transducer Interface Draft Standards. In: SENSORS EXPO, 1996, Philadelphia. **Proceedings of the...**, Philadelphia: Helmers Publishing, 1996, p.87 – 100.

[21] CUMMINS, T.; BRANNICK, D.; BYRNE, E.; O´MARA, B.; STAPLETON, H.; CLEARY, J.; O'RIORDAN, J.; LYNCH, D.; NOONAN, L.; DEMPSEY, D. An IEEE 1451 standard transducer interface chip. In: INTERNATIONAL SOLID-STATE CIRCUITS CONFERENCE - ISSCC, 45, 1998. **Conference....**IEEE, 1998. p.276 - 277.

[22] CUMMINS, T.; BYRNE, E.; BRANNICK, D.; DEMPSEY, D. An IEEE 1451 Standard transducer interface chip with 12-b ADC, two 12-b DAC's, 10-kB Flash EEPROM, and 8-b microcontroller. **IEEE Journal of Solid-State Circuits**, IEEE, v.33, no. 12, p.2112 - 2120, Dec. 1998.

[23] CONWAY, P.; HEFFERNAN, D.; O'MARA, B.; BURTON, P.; MIAO, T. IEEE 1451.2: an interpretation and example implementation. In: INSTRUMENTATION AND MEASUREMENT TECHNOLOGY CONFERENCE - IMTC, 17, 2000, Baltimore. **Proceedings of the...**, Baltimore: IEEE, 2000, v.2, p. 535 - 540.

[24] ANALOG DEVICES. The ADuC812 MicroConverter as an IEEE 1451.2 compatible smart transducer interface. Application Note, MicroConverter™ technical note - uC003, Ver. 1.0, Sept. 1999. Disponível em: <**www.analog.com/microconverter**>. Acesso em: 2002.

[25] CAMARA, L.; RUIZ, O.; SAMITIER, J. Complete IEEE-1451 node, STIM and NCAP, implemented for a CAN network. In: INSTRUMENTATION AND MEASUREMENT TECHNOLOGY CONFERENCE, IMTC, 17, 2000, Baltimore. **Proceedings of the...**, Baltimore: IEEE, 2000. v. 2, p.541- 545.

[26] GIRERD, C.; GARDIEN, S.; BURCH, J.; KATSANEVAS, S.; MARTEAU, J. Ethernet based-network DAQ and smart sensors for the OPERA long-baseline neutrino experiment. In: NUCLEAR SCIENCE SYMPOSIUM CONFERENCE RECORD, 2000. **Proceedings of the...**, IEEE, 2000. v. 2, p.111-115.

[27] WOBSCHALL, D. An implementation of IEEE 1451 NCAP for internet access of serial port-based sensors. In: SENSORS FOR INDUSTRY CONFERENCE, 2, 2002. **Conference...**.IEEE, 2002. p.157- 160.

[28] ESENSORS. IEEE 1451 NCAP interface with multiple serial ports. Technical Specification, EM04a. Disponível em: <**www.eesensors.com**>. Acesso em: 2003.

[29] CASTRO A.; RIESGO, T.; de la TORRE, E.; TORROJA, Y.; UCEDA, J. Custom hardware IEEE 1451.2 implementation for smart transducers. In: INDUSTRIAL ELECTRONICS CONFERENCE - IECON, 28, 2002. **Proceedings of the...**, IEEE, 2002, v.4, p.2752-2757.

[30] CASTRO, A.; CHAQUET, J. M.; MOREJON, E.; RIESGO, T.; UCEDA, J. A system- on-chip for smart sensors. In: INTERNATIONAL SYMPOSIUM INDUSTRIAL ELECTRONICS - ISIE, 2002. **Proceedings of the...**, IEEE, 2002. v. 2 , p. 595 –599.

[31] KOCHAN, R.; LEE, K.; KOCHAN, V.; SACHENKO, A. Development of a dynamically reprogrammable NCAP. In: INSTRUMENTATION AND MEASUREMENT

TECHNOLOGY CONFERENCE - IMTC, 21, 2004, Como (Italy). **Proceedings of the...**, Como: IEEE, 2004, v.2, p. 1188 – 1193.

[32] YANFENG, W.; MASAKATSU, N.; MIKIKO, Y.; MAKOTO, N.; MASAICHI, F.; KENZO, W. A NDIR CO_2 monitor with smart interface for global networking. In: INSTRUMENTATION AND MEASUREMENT TECHNOLOGY CONFERENCE - IMTC, 21, 2004, Como (Italy). **Proceedings of the...**,Como: IEEE, 2004, v.2, p. 1194 - 1198.

[33] TANENBAUM, A. S. **Redes de Computadores**. 3 ed. Rio de Janeiro: Campus, 1997. 884 p.

[34] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, Computer Society. **IEEE Standard for Local Area Networks - Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications (Std. 802.3)**. IEEE. NY: ANSI/IEEE, 1985. 146 p.

[35] TORRES, G. **Redes de Computadores: Curso Completo**. Rio de Janeiro: Axcel Books do Brasil, 2001. 664 p.

[36] MARINO, A. F. A Internet na automação industrial. Boletim Engecomp, no. 28, Maio 2000. Disponível em: <**www.engecomp.com.br/bol_0028.htm**> Acesso em: 2002.

[37] YAMASAKI, H. **Intelligent Sensors**. Serie: Handbook of Sensors and Actuators, v. 3. Amsterdam, NY: Elsevier, 1996. 297 p.

[38] SZE, S. M. **Semiconductor Sensors**. NY: John Wiley & Sons, Inc., 1994. 550 p.

[39] PALLÁS-ARENY, R.; WEBSTER, J. G**. Sensors and Signal Conditioning**. 1 ed. NY: John Wiley & Sons, Inc., 1991. 398 p.

[40] GARDNER, J. W. **Microsensors Principles and Applications**. NY: John Wiley & Sons, Inc., 1994. 331 p.

[41] CHONG, C. Y.; KUMAR, S. P. Sensor networks: evolution, opportunities and challenges **Proceedings of the IEEE**, IEEE, v.91, no. 8, p. 1247-1256, Aug. 2003.

[42] WARRIOR, J. Smart sensor networks of the future. Sensors Magazine, 1997. Disponível em: <**www.sensorsmag.com/articles/1097/ieee1097/main.shtml**>. Acesso em: 2001.

[43] PASSOS PEREIRA, A. Como implementar proyectos con Foundation Fieldbus. Tutorial. Disponível em: <**www.ineco.cl/centralmarco.htm**>. Acesso em: 2003.

[44] KESTER, W.; CHESTNUT, B.; KING, G. Smart Sensors. Papers in PDF format. Disponível em : <**www.analog.com/technology/amplifiersLinear/training/pdf/Sensor_**

sect9.pdf>. Acesso em: 2002.

[45] NAJAFI, N.; WISE, K. D. An organization and interface for sensor-driven semiconductor process control systems. **IEEE Trans. Semiconductor Manufacturing**, IEEE, v.3, no. 4, p.230 - 238, Nov. 1990.

[46] CORREIA, H.; CRETU, E.; BARTEK, M.; WOLFFENBUTTEL, R. F. A microinstrumentation system for industrial applications. In: INTERNATIONAL SYMPOSIUM INDUSTRIAL ELECTRONICS - ISIE, 1997. **Proceedings of the...**, IEEE, 1997, p. 846 - 840.

[47] ACTUATOR SENSOR INTERFACE. ASI site. Disponível em: <**www.as-interface.com**>. Acesso em: 2003/2004.

[48] CAN DATALINK LAYER. Tutorial. Disponível: <**www.datamicro.ru/can/standards /CANdll.pdf**>. Acesso em: 2001.

[49] CAN IN AUTOMATION. CIA Group website. Disponível em: <**www.can-cia.de/can**>. Acesso em: 2003/2004.

[50] CONSUMER ELECTRONIC BUS. CEbus website. Disponível em: <**www.cebus.org**>. Acesso em: 2003/2004.

[51] OPEN DEVICENET VENDORS ASSOCIATION. ODVA website. Disponível em: <**www.devicenet.org**>. Acesso em: 2003/2004.

[52] FOUNDATION FIELDBUS. Foundation Fieldbus website. Disponível em <**www.fieldbus.org**>. Acesso em: 2002/2003/2004.

[53] HART COMMUNICATION FOUNDATION. HART Communication Foundation website. Disponível em: <**www.hartcomm.org**>. Acesso em: 2003/2004.

[54] INTERBUS. Interbus website. Disponível em: <**www.interbus.com**>. Acesso em: 2003.

[55] ECHELON. LonWorks website. Disponível em: <**www.echelon.com/products/lonworks/default.htm**>. Acesso em: 2003/2004.

[56] PROFIBUS. Profibus website. Disponível em: <**www.profibus.com**>. Acesso em: 2003.

[57] UNIVERSITAT D' VALENCIA. Redes de comunicación industriales. Papers in PDF Format. Disponível em: <**http://juce.galeon.com/artredind.pdf**>. Acesso em: 2003.

[58] HONEYWELL MICROSWITCH. SDS website. Disponível em: <**http://content.honeywell.com/sensing/prodinfo/sds**>. Acesso em 2003/2004.

[59] SERCOS. Interest Group SERCOS-Interface website. Disponível em: <**www.sercos.org**> Acesso em: 2003/2004.

[60] WORLDFIP. WorldFIP website. Disponível em: <**www.worldfip.org**>. Acesso em: 2003/2004 .

[61] LEE, K.B. A synopsis of the IEEE P1451- Standards for smart transducer communication. Papers in PDF format, National Institute of Standards and Technology. Disponível em: <**http://ieee1451.nist.gov**>. Acesso em: 2002.

[62] TELEMONITOR INC. IEEE 1451 website. Disponível em: <**www.telemonitor.com/ieee 1451.html**>. Acesso em 2001/2002/2003/2004.

[63] FISCHER, R.; BURCH, J. The PICmicro® MCU as an IEEE 1451.2 compatible Smart Transducer Interface Module (STIM). Application Note, AN214 - Microchip Technology, 2000. Disponível em: <:**http://www.microchip.com/1010/suppdoc/appnote/alpha**>. Acesso em: 2002.

[64] JOHNSON, R. N.; WOODS, S. P. Overview and status update for IEEE1451.2 transducer to microprocessor communications protocols and Transducer Electronic Data Sheet (TEDS) formats. Papers in PDF format, Telemonitor Inc. Disponível em: <**www.telemonitor.com/doc/dot2.pdf**>. Acesso em: 2002.

[65] WOODS, S. P.; BRYZEK, J.; CHEN, S.; CRANMER, J.; EL-KAREH, E. V.; GEIPEL, M.; GEN-KUONG, F.; HOULDSWORTH, J.; LeCOMTE, N.; LEE, K.; MATTES, M. F.; RASMUSSEN, D. E. IEEE - P1451.2 smart transducer interface module. In: SENSORS EXPO, 1996, Philadelphia. **Proceedings of the...**, Philadelphia: 1996, p. 25 – 38.

[66] SMITH, R.D. Building IEEE 1451.2 Smart Transducer Interface Modules (STIMs). In: SENSORS EXPO, 2000, Anaheim. PPT Format.

[67] TOCCI, R. J. **Microprocessadores e Microcomputadores: Hardware e Software**. 3 ed. Rio de Janeiro: Prentice-Hall do Brasil, 1990. 346 p.

[68] SCHNEEMAN, R. An IEEE 1451.1 Summary. National Institute of Standards and Technology, 2000. Disponível em: <**http://ieee1451.nist.gov/Workshop_04Oct01/HTML_version/Schneeman/Schneeman_01.html**>. Acesso em 2002.

[69] SPURGEON's, C. Ethernet Tutorial. Disponível em: <**www.ethermanage.com/ethernet**>. Acesso em: 2002/2003/2004.

[70] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. IEEE 802.3 (CSMA/CD) website. Disponível em: <**http://grouper.ieee.org/groups/802/3**>. Acesso em: 2004.

[71] BEUERMAN, S. L.; COYLE, J. E. The delay characteristics of CSMA/CD networks. **IEEE Trans. on Communications**, IEEE, v.36, no. 5, p.553-563, May 1988.

[72] MOLLE, M. L. A new binary logarithmic arbitration method for Ethernet. Technical Report CS-RI 298. Papers in PDF format, April 1994. Disponível em: <**www.ethermanage.com/ethernet/pdf/molle-report.pdf**>. Acesso em: 2004.

[73] FARIA, C. G.; CARNEIRO, C. M.; HEILGENDORFF, R. M. Development and implementation of a supervision and control system for the plant of gas treatment of the anode bucking furnace at Albras. In: CONFERÊNCIA INTERNACIONAL DE APLCAÇÕES INDUSTRIAIS – INDUSCON, 6, 2004. **Proceedings of the...**, Joinville, 2004, p.98-93.

[74] COSTA, R. P.; NETO, J. C. O uso de redes Ethernet com diferenciação de tráfego em aplicações industriais. In. CONGRESSO BRASILEIRO DE AUTOMÁTICA – CBA, 15, 2004. **Proceedings of the…**, Gramado, 2004, Código: 970.

[75] BROWN, S.; ROSE, J. Architecture of FPGAs and CPLDs: a tutorial. Papers in PDF format, Department of Electrical and Computer Engineering University of Toronto. Disponível em: <**http://ccc.inaoep.mx/~rcumplido/referencia/Architecture%20%of%20FPGAS%20CPLDs-%20A%20Tutorial.pdf**>. Acesso em: 2004.

[76] ALTERA. Flex 8000 Handbook. Altera Corporation, 1994. 233 p.

[77] UNIVERSIDAD DE ZARAGOZA. Circuitos Lógicos Programables, arquitectura. Tutorial. Dpto. de Ingeniería Electrónica y Comunicaciones. Disponível em: <**www.cps.unizar.es/~te/Docencia_ archivos/ldh_archivos/T1.pdf**>. Acesso em 2003.

[78] RIESGO, T.; TORROJA, Y.; de la TORRE, E. Design methodologies based on hardware description languages. **IEEE Trans. Industrial Electronics**, IEEE, v.46, no. 1, p.3-12, Feb. 1999.

[79] ZORAN, S.; ASSIM, S. **Digital Systems Design and Prototyping – Using Field Programmable Logic and Hardware Description Languages**. 2 ed. Kluer Academics, 2000. 620 p.

[80] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, Computer Society. **IEEE Standard VHDL - Language Reference Manual. (Std. 1076-1987)**. IEEE. NY: IEEE, 1987. 218 p.

[81] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, Computer Society. **IEEE Standard VHDL - Language Reference Manual. (Std. 1076-1993)**. Standards Board. NY: IEEE, 1993. 249 p.

[82] PERRY, D. L. **VHDL**. 2 ed. NY: McGraw-Hill, 1994. 390 p.

[83] ALTERA. Max+Plus II Getting Started. Tutorial. Altera Corporation, 1997. 353 p.

[84] ALTERA. Quartus Tutorial. Tutorial. Altera Corporation, Versão 1999.06, 1999. 109 p.

[85] UNIVERSIDAD DE BURGOS. Guía de iniciación al lenguaje Java. Tutorial. Ver. 2.0, Nov. 1999. Disponível em: <**http://pisuerga.inf.ubu.es/lsi/Invest/Java/Tuto/Index.htm**> Acesso em: 2003.

[86] MURTA, L.G.P.; WERNER, C.M.L.; OLIVEIRA BARROS, M. Café da manhã com Java. Relatório Técnico. Universidade Federal do Rio de Janeiro, Nov.1998. Disponível em: < **https://sety.cos.ufrj.br/prometeus/publicacoes/ java.pdf?PHPSESSID=3731320cc62bde356c385cca066b329c0**>. Acesso em: 2003.

[87] BOOCH, G. **Object-Oriented Analysis and Design**. 2 ed. Addison-Wesley Publishing Company, 1994. 589 p.

[88] SUN MICROSYSTEMS. Sun website. Disponível em: <**www.java.sun.com**>. Acesso em: 2003/2004 .

[89] BACLAWSKY, K. Java RMI Tutorial. Tutorial. Disponível em: <**http://www.ccs.neu.edu/home/kenb/com3337/rmi_tut.html**>. Acesso em 2003.

[90] SUN MICROSYSTEMS. Java Native Interface website. Disponível em: <**http://java.sun.com/docs/books/tutorial/native1.1/**>. Acesso em 2003/2004.

[91] DEL CID PORTILLO, J. Parallel printer port access through Java. Disponível em:<**http://www.geocities.com/Juanga69/Parport/**>. Acesso em 2003/2004.

[92] MECATRÔNICA FÁCIL. Mecatrônica Fácil website. Utilizando o LOGO em Windows 2000, NT e XP. Disponível em: <**http://www.mecatronicafacil.com.br/downloads/logo_instr.htm** >. Acesso em 2004.

[93] MAHMOUD, Q.H. Sockets programming in Java: a tutorial. Disponível em: <**http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets-p2.html>.** Acesso em 2004.

[94] MANGIONE, C. Performance tests show Java as fast as C++. Disponível em: <**http://www.javaworld.com/javaworld/jw-02-1998/jw-02-jperf_p.html>**. Acesso em 2004.

[95] CENTER OF DISTRIBUTED OBJECT COMPUTING (DOC). Java RT website. Disponível em: < **http://tao.doc.wustl.edu/rtj** >. Acesso em 2004.

[96] BOLLELLA, R.; DIBBLE, B.B.P.; GOSLING, J.S.F.; TURNBULL, D.H.M.; BELLIARDI, R. The real-time specification for Java™. Especificação Técnica, 2000. Disponível em: <**https://rtsj.dev.java.net/**>. Acesso em 2004.

[97] JOHNSON, R. N. Building plug-and-play networked smart transducers. Sensors Magazine, 1997.Disponível em: <**wysiwyg://main.9/http://www.sensorsmag.com/articles/1097/ieee1097/main.shtml**>. Acesso em: 2001.

[98] ALTERA. Max+Plus II VHDL. Altera Corporation. Databook, ver. 7.1, 1996. 233 p.

[99] GUGEL, K.; SCHWARTZ, E, M. MaxPlus II ROM creation instructions. Tutorial. Department of Electrical & Computer Engineering, University of Florida, Nov 2001. Disponível em: <**www.mil.ufl.edu/eel3701/docs/MaxPlus/ROM_Creation.pdf**>. Acesso em: 2002.

[100] ALTERA. University Program Design Laboratory Package. Altera Corporation, Ver.1, 1997. 28 p.

[101] ALTERA CORPORATION. Altera Devices. Disponível em: <**www.altera.com/products/devices/dev-index.jsp**>. Acesso em 2003/2004.

[102] ALTERA. Acex 1K Programmable Logic Device Family. Altera Corporation. Data Sheet, ver. 3.3, 2001.

[103] ZELENOVSKY, R.; MENDONÇA, A. **Hardware e interfaceamento**. 3 ed. Rio de Janeiro: MZ, 2002. 1032 p.

[104] BATISTA, E. A.; ROSSI, S. R.; SILVA, A. C. R.; CARVALHO, A. A. KITANO, C. Proposta de implementação de um sistema de monitoramento remoto baseado no padrão IEEE1451. In: WORLD CONGRESS ON ENGINEERING AND TECHNOLOGY EDUCATION - WCETE, 19, 2004, Santos. **Proceedings of the...**, Santos. 2004, p. 787 - 791.

[105] BATISTA, E.A. **Emprego da Tecnologia Java para Implementar a Parte Lógica de um Processador de Aplicação com Capacidade de Operar em Rede de Comunicação (NCAP), em Conformidade com o Padrão IEEE 1451**. Dissertação (mestrado). UNIVERSIDADE ESTADUAL PAULISTA, FACULDADE DE ENGENHARIA, FE/IS-UNESP. Ilha Solteira: [s.n], 2004.140 p.

[106] DE JESUS, M.R.; BATISTA, E. A.; ROSSI, S. R.; SILVA, A. C. R.; CARVALHO, A. A. Implementação de um website dinâmico e interativo para controlar e monitorar transdutores inteligentes via Internet, conforme o padrão IEEE1451. In: GLOBAL CONGRESS ON ENGINEERING AND TECHNOLOGY EDUCATION – GCETE, 2005. Proceed..., Bertioga, art.: 571.

GLOSSÁRIO

Atuador: dispositivo que converte um sinal elétrico em uma ação física.

Barramento de campo: sistema de interconexão digital bidirecional para dispositivos transdutores (sensores e atuadores), que sob um determinado protocolo, possibilita a troca de informação entre eles.

***Bit*:** abreviação de *binary digit* (dígito binário).

***Byte*:** conjunto de 8 *bits*.

***Buffer*:** elemento intermediário para armazenamento temporário

Complemento de 1: resultado obtido ao se tomar um número binário e converter cada *bit* em seu oposto.

Condicionamento de sinal: processo de amplificação, compensação, filtragem, etc, que se efetua sobre o sinal vindo de um sensor ou indo para um atuador.

Disparo: sinal que estabelece o começo de um evento. Como definido no IEEE 1451.2, um disparo é um sinal desde o NCAP para o STIM servindo como um comando para uma ação determinada ocorrer.

EEPROM (PROM eletricamente apagável): dispositivo de memória ROM que permite o apagamento e a reprogramação por parte do usuário

Endereço completo: combinação de um endereço de função e um endereço de canal. O endereço completo especifica se o dado é do tipo leitura ou escrita, a função e o canal.

Endereço de Canal: parte de um *endereço completo* que especifica o canal que é lido ou escrito.

Endereço de função: parte de um *endereço completo* que especifica o tipo de função a ser executada.

***Firmware*:** programas e dados armazenados em memória ROM e responsáveis pela iniciação de um processador e seus circuitos de interface.

***Half-duplex*:** tipo de comunicação serial onde os dados podem ser transmitidos e recebidos, mas não simultaneamente.

Handshake: processo de estabelecimento de um contato inicial entre dois dispositivos de comunicação.

***Host**:* termo utilizado para fazer referência a um sistema computador conectado a uma rede.

Hot-swap: termo que faz referência ao processo de conectar ou desconectar o STIM, sem desligar a alimentação que o NCAP lhe fornece através da interface TII.

Interface: limite comum entre duas camadas ou subsistemas, que define as operações e serviços que uma camada tem a oferecer para a outra.

Interoperabilidade: possibilidade de conectar com sucesso, dispositivos de diferentes fornecedores em um ambiente determinado.

Interrupção: sinal de entrada a um processador que se utiliza para suspender um processo e começar outro. No padrão IEEE 1451.2, uma interrupção é um sinal utilizado pelo módulo STIM para requisitar um serviço ao NCAP.

Interrupção mascarável: tipo de interrupção que permite ignorar a requisição de interrupção, se o sinalizador correspondente estiver ativado.

Interrupção não mascarável: tipo de interrupção que força a unidade microprocessadora a responder à requisição, independentemente do estado do sinalizador de inibição.

Meta: prefixo de origem grega que significa aquele que pertence a uma entidade global ou que possui alguma coisa em comum o em combinação com todos os membros que formam um conjunto.

NCAP (Processador de Aplicação com Capacidade de Operar em Rede): dispositivo intermediário entre o módulo STIM e a rede. Executa comunicações com uma rede, comunicações com o STIM e funções de conversão de dados.

Operação de escrita: transferência de dados (*bits*) desde o módulo NCAP para o modulo STIM.

Operação de leitura: transferência de dados (*bits*) desde o módulo STIM para o modulo NCAP.

PLD (*Programmable Logic Device*): circuito integrado configurável pelo usuário final e programável via *software*, utilizado para implementar funções lógicas envolvidas em sistemas digitais de complexidade variável

Plug and Play: termo derivado das arquiteturas de computadores *plug and play*. Significa conecte e opere, fazendo referência aos sistemas flexíveis, nos quais é possível acrescentar dispositivos sem ter que executar tarefas de atualização e reconfiguração.

Protocolo: conjunto de regras sobre o modo como se estabelecerá a comunicação entre as partes envolvida em um sistema.

Quadro de escrita: estrutura de dados (*bits*) utilizada em uma operação de escrita.

Quadro de leitura: estrutura de dados (*bits*) utilizada em uma operação de leitura.

Reconhecimento (*acknowledgment*): sinal que é utilizado para responder a uma mensagem, que sua informação ou sinal foi recebido.

Registrador: grupo de dispositivos de memória (biestáveis) utilizados para armazenar informação binária.

ROM (Memória exclusivamente de leitura): dispositivo de memória, semicondutor, projetado para que os dados sejam gravados apenas uma vez e, depois, lidos inúmeras vezes.

Sensor: dispositivo que converte um sinal de natureza física, biológica ou química, em um sinal de tipo elétrico.

STIM (Modulo Interface para Transdutores Inteligentes): módulo que contém a informação dos transdutores em formato eletrônico (TEDS), a lógica para implementar a interface para transdutores inteligentes, o transdutor(es) e qualquer bloco de conversão de sinal ou condicionamento associado.

TEDS (Especificações de transdutores em formato eletrônico): "folha de dados" que caracteriza um transdutor, armazenada eletricamente num dispositivo de memória do tipo apenas leitura.

Tempo de *setup*: tempo que transcorre desde que se produz a requisição inicial para executar uma função até o inicio da atividade solicitada.

TII (Interface Independente de Transdutores): interface digital padronizada, utilizada para conectar o módulo STIM com o NCAP.

Transdutor: dispositivo capaz de converter energia de um domínio para outro, na mesma ou em diferente forma.

Transdutor inteligente: transdutor que fornece outras funções além das necessárias, a fim gerar uma representação correta de uma quantidade medida ou controlada. Esta funcionalidade simplifica a integração dos transdutores em ambientes de rede.

APÊNDICES

APÊNDICE A – Grupos de Trabalho IEEE 802

A seguir apresenta-se uma relação das principais diretrizes da família IEEE 802, sobre padronização de tecnologia de LANs:

- IEEE 802.1: Relação entre as especificações da família 802 e sua relação com o modelo ISO/OSI.
- IEEE 802.2: Controle de Enlace Lógico (LLC).
- IEEE 802.3: CSMA/CD (*Ethernet*).
    - IEEE 802.3u: *Fast Ethernet*.
    - IEEE 802.3z (1998): *Gigabit Ethernet*.
    - IEEE 802.3ab (1999): *Gigabit Ethernet* 1000 BASE-T.
    - IEEE 802.3ae (2002): 10 *Gigabit Ethernet*.
    - IEEE 802.3ak (2004): 10GBASE-CX4.
- IEEE 802.4: Tecnologia *Token Bus*.
- IEEE 802.5: Tecnologia *Token Ring*.
- IEEE 802.6: Redes WAN.
- IEEE 802.10: Segurança em LANs.
- IEEE 802.11: Redes LAN sem fio.
- IEEE 802.15: Redes de Área Pessoal sem Fio (WPAN - *Wireless Personal Area Network*).
- IEEE 802.16: Acesso a Redes sem Fio de Banda Ampla.

APÊNDICE B – Estado Atual do Padrão IEEE 1451

B1 - IEEE Std 1451.1-1999

Nome oficial: *Standard for a Smart Transducer Interface for Sensors and Actuators - Network Capable Application Processor (NCAP) Information Model for Smart Transducers*.
Estado: Aprovado.
Características: Nesta especificação descrevem-se classes, métodos e procedimentos relacionados com a conexão de transdutores inteligentes em rede.

B2 -IEEE Std 1451.2-1997

Nome oficial: *Standard for a Smart Transducer Interface for Sensors and Actuators - Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats.*
Estado: Aprovado.
Características: Nesta especificação descreve-se a interface de *hardware* e os protocolos de comunicação entre o STIM e o NCAP. A diretriz inclui a definição das especificações dos transdutores em formato eletrônico (TEDS).

B3 - IEEE Std 1451.3-2003

Nome oficial: S*tandard for a Smart Transducer Interface for Sensors and Actuators - Digital Communication and Transducer Electronic Data Sheet (TEDS) Formats for Distributed Multidrop Systems.*
Estado: Aprovado.
Características: Na especificação descreve-se a forma de comunicação para um arranjo de vários transdutores de forma distribuída, cujas informações precisam ser lidas de forma sincronizada.

B4 - IEEE Std 1451.4-2004

Nome oficial: S*tandard for a Smart Transducer Interface for Sensors and Actuators - Mixed-Mode Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats.*
Estado: Aprovado.

Características: O modo misto possibilita que um modo analógico, por exemplo, o sinal de um transdutor, e um modo digital, por exemplo, um dado contido em memória, possam ser disponibilizados através da mesma interface com o NCAP. Este fato é importante para aplicações que requerem de elevadas taxas de transferência de dados.

B5 - IEEE P1451.5

Nome oficial: *Draft Standard for a Smart Transducer Interface for Sensors and Actuators - Wireless Communication Protocols and Transducer Electronic Data Sheets*.
Estado: Especificação proposta, trabalho em andamento.
Características: A idéia deste grupo é criar uma especificação para direcionar as aplicações do padrão IEEE 1451, em ambientes de rede sem fio.

B6 - IEEE P1451.0

Estado: Especificação proposta recentemente.
Características: A idéia do grupo IEEEE P1451.0 é criar uma especificação que serva como base para todas as outras especificações da família 1451, por exemplo, a unificação das TEDS das especificações componentes.

B7 - IEEE P1451.6

Nome: *A High-speed CANopen-based Transducer Network Interface for Intrinsically Safe and Non-intrinsically Safe Applications*
Estado: Especificação proposta recentemente.
Características: Neste caso propõe-se o uso de uma rede de alta velocidade baseada no sistema *CAN-open* com diversos módulos contendo transdutores e a definição de uma camada de segurança no modelo de comunicação.

APÊNDICE C - Endereços de Função IEEE 1451.2

Tabela C1. Endereços de função, globais.

Address	Function
0	Write global transducer data
1	Write global control command
3	Write triggered channel address
5	Write global standard interrupt mask
96	Write global End-User's Application -Specific TEDS
128	Read global transducer data
130	Read global standard status
132	Read global auxiliary status
160	Reda Meta-TEDS
161	Read Meta-Identification TEDS
224	Read global End-User's Application -Specific TEDS

Fonte: Std. IEEE 1451.2.

Tabela C2. Endereços de função para os canais 1 a 255.

Address	Function
0	Write channel transducer data
1	Write channel control command
5	Write channel standard interrupt mask
64	Write Calibration TEDS
65	Write Calibration Identification TEDS
96	Write End-User's Application -Specific TEDS
128	Read channel transducer data
130	Read channel standard status
132	Red channel auxiliary status
160	Read Channel TEDS
161	Read Channel Identification TEDS
192	Read Calibration TEDS
192	Read Calibration Identification TEDS
224	Read End-User's Application -Specific TEDS

Fonte: Std. IEEE 1451.2.

APÊNDICE D – Instalação do JDK, *Parport* e *Userport*

D1: Instalação do *Java Development Kit* (JDK)

No *site* da *Sun Microsystems* [88] são disponibilizadas as seguintes plataformas:

a) J2ME: *Java 2 Micro Edition* – celular, *palmtop*, etc;

b) J2EE: Java 2 *Enterprise Edition* – servidores;

c) J2SE: *Java 2 Standard Edition* – estações de trabalho.

Para a aplicação desenvolvida neste trabalho, empregou-se o *kit* JDK, versão 1.4.0, pertencente à plataforma J2SE.

A fim de instalar o JDK, no *site* <http://www.java.sun.com/j2se> deve ser escolhida a versão do JDK, por exemplo, 1.4.0. Logo depois, deve ser descarregada de acordo com o sistema operacional utilizado.

No processo de instalação copie o JDK em um diretório adequado, por exemplo: c:\j2sdk1.4.0.

Antes de executar o JDK, o caminho adequado deve ser criado, incluindo no arquivo *autoexec*, as variáveis de ambiente *path* e *classpath* (*Windows*).

```
set path=%path%;c:\j2sdk1.4.0\bin
set classpath=.;c:\j2sdk1.4.0\lib
```

Finalmente, o microcomputador deve ser reiniciado.

Notar-se-á que dentro do diretório j2sdk1.4.0 irão aparecer várias pastas criadas no processo de instalação, algumas dessas pastas são:

bin: contém os programas necessários à compilação e execução, como o *javac* e o *rmic*;

jre: contém os programas e as bibliotecas necessárias à execução de códigos em Java;

include: contém interfaces para programas em C, C++ e *assembly*;

lib: bibliotecas.

D2: Instalação do Pacote *Parport*

a) Descarregar o arquivo *parport-win32.zip* do *site*: http://www.geocities.com/Juanga69/Parport;

b) Descompactar o arquivo num diretório do seu PC, por exemplo: C:\j2sdk1.4.0\parport. Dentro da pasta criada serão disponibilizados: o arquivo *parport.dll*, os códigos fonte em C, alguns exemplos e um tutorial básico[44];

c) Copiar o arquivo *parport.dll* em C:\j2sdk1.4.0\bin;

d) O caminho adequado deve ser criado, incluindo no arquivo *autoexec*, a variável de ambiente *classpath* (*Windows*): set CLASSPATH=c:\j2sdk1.4.0;%CLASSPATH%;

e) Reinicie o seu computador. O sistema ficará pronto para executar programas que utilizam a porta paralela do PC, para operações de escrita e leitura;

e) Caso o sistema operacional usado seja *Windows® NT/2000/XP*, o aplicativo *Useport* deve ser instalado a fim de liberar a porta paralela para usuários.

D3: Instalação do Aplicativo *Userport*

a) Descarregar o arquivo *userport.zip* do *site*: http://www.geocities.com/Juanga69/Parport;

b) Descompactar o arquivo num diretório do seu PC, por exemplo: C:\j2sdk1.4.0\userport;

c) Copiar o arquivo *UserPort.sys* em ....\System32\Drivers;

d) Executar o programa e configurá-lo de acordo com a seção 6.3.6 b).

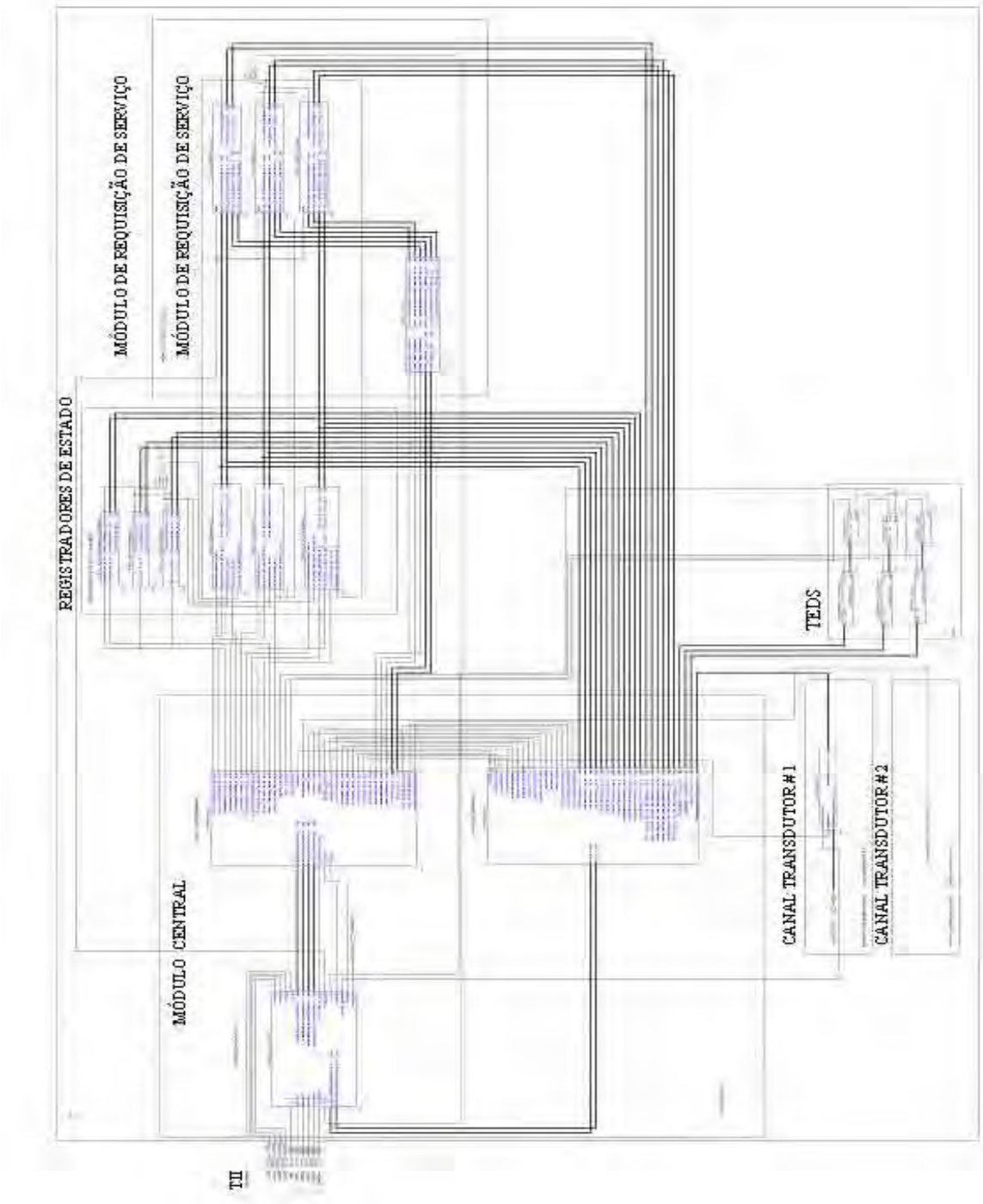
D4: Instalação da API de Comunicações

A fim de instalar a API *comm* corretamente, pode-se acessar o *site* da *Sun Micrsystems* [88], porém, recomenda-se o material disponível em: www.ufsm.br/gmicro/porta_serial_java.htm, o qual traz um excelente guia de instalação tanto para *Windows* quanto para *Linux*.

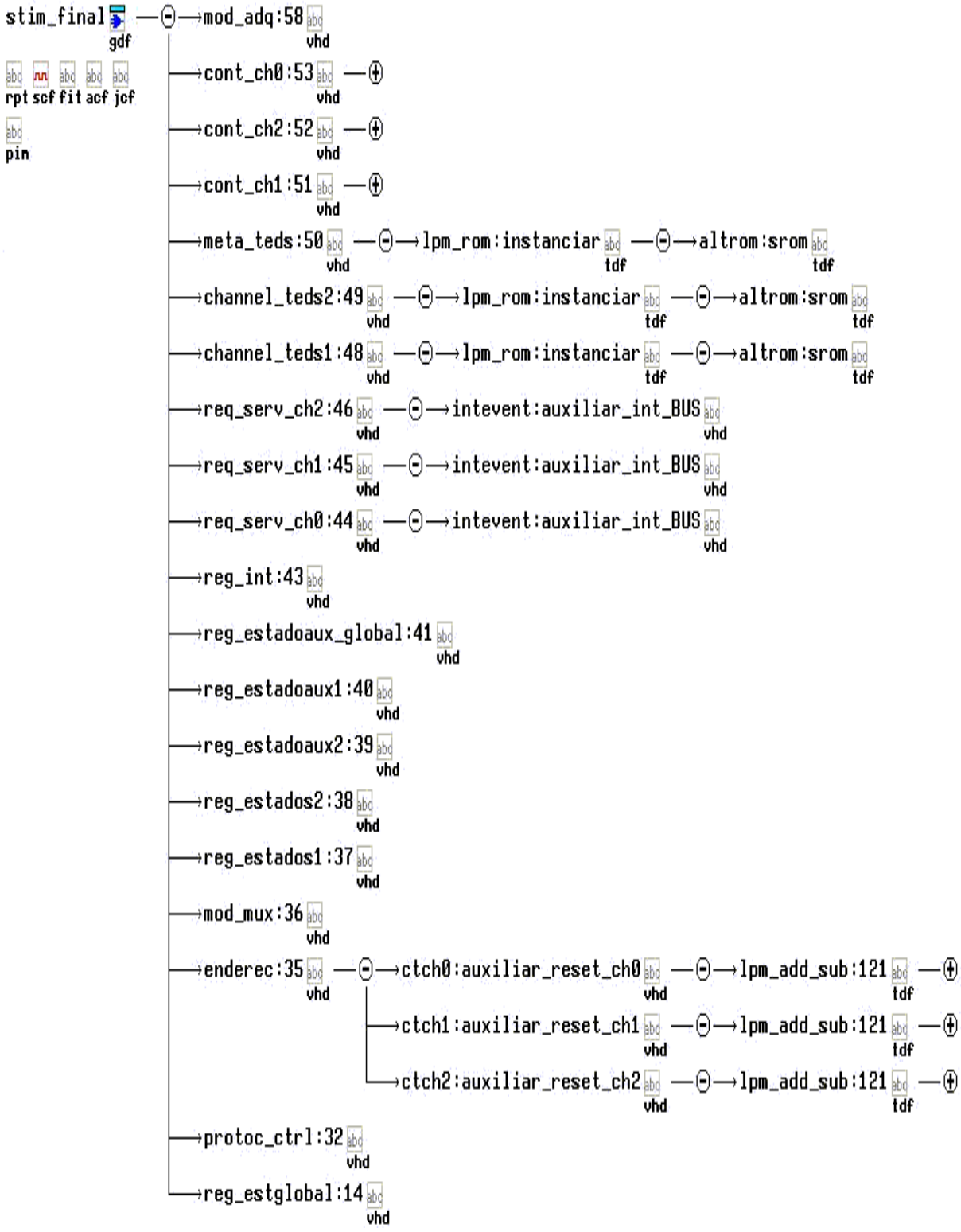
[44] No *site* também está disponível uma versão para *Linux* com tutorial.

APÊNDICE E – Arquiteturas e Hierarquias de Projeto

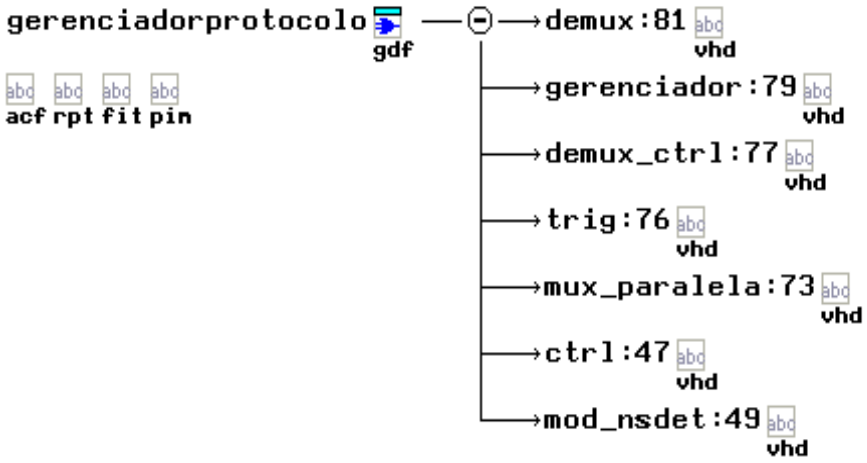
E1 – Arquitetura do STIM



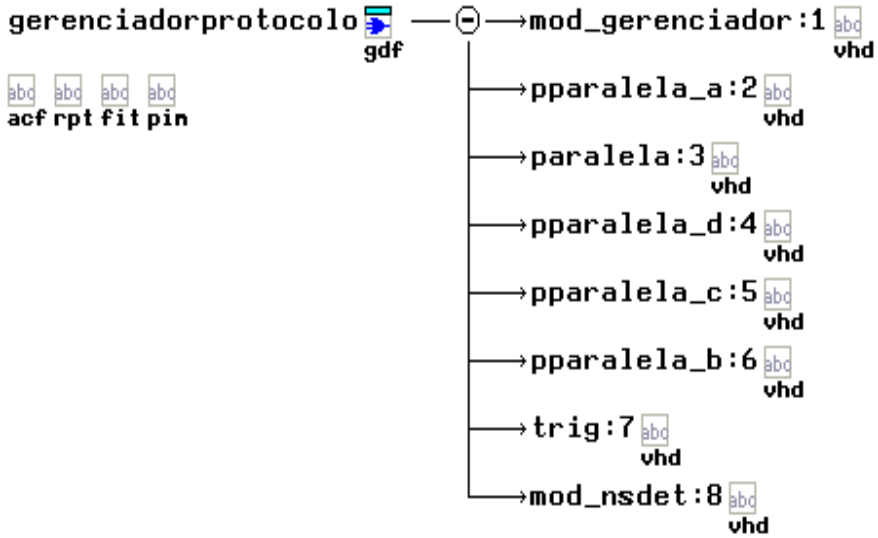
E1.1 – Hierarquia de Projeto do STIM



E2 – Hierarquia de Projeto do Gerenciador de Protocolo com Entrada e Saída de 8 *Bits*



E3 – Hierarquia de Projeto do Gerenciador de Protocolo com Entrada de 8 *Bits* e saída de 5 *Bits*



APÊNDICE F – Detalhes dos Circuitos de Condicionamento e Conversão de Sinal

F1: STIM Implementado: Circuitos de Condicionamento e Conversão de Sinal

Na Figura F1 é mostrado o diagrama dos circuitos necessários para condicionar o sinal fornecido pelo sensor LM35 conectado ao canal 1. Utilizou-se um amplificador operacional OP77 de baixo *offset*, um regulador de tensão integrado modelo LM7805, um conversor A/D de 8 *bits* modelo AD0804 de aproximações sucessivas, e acessórios.

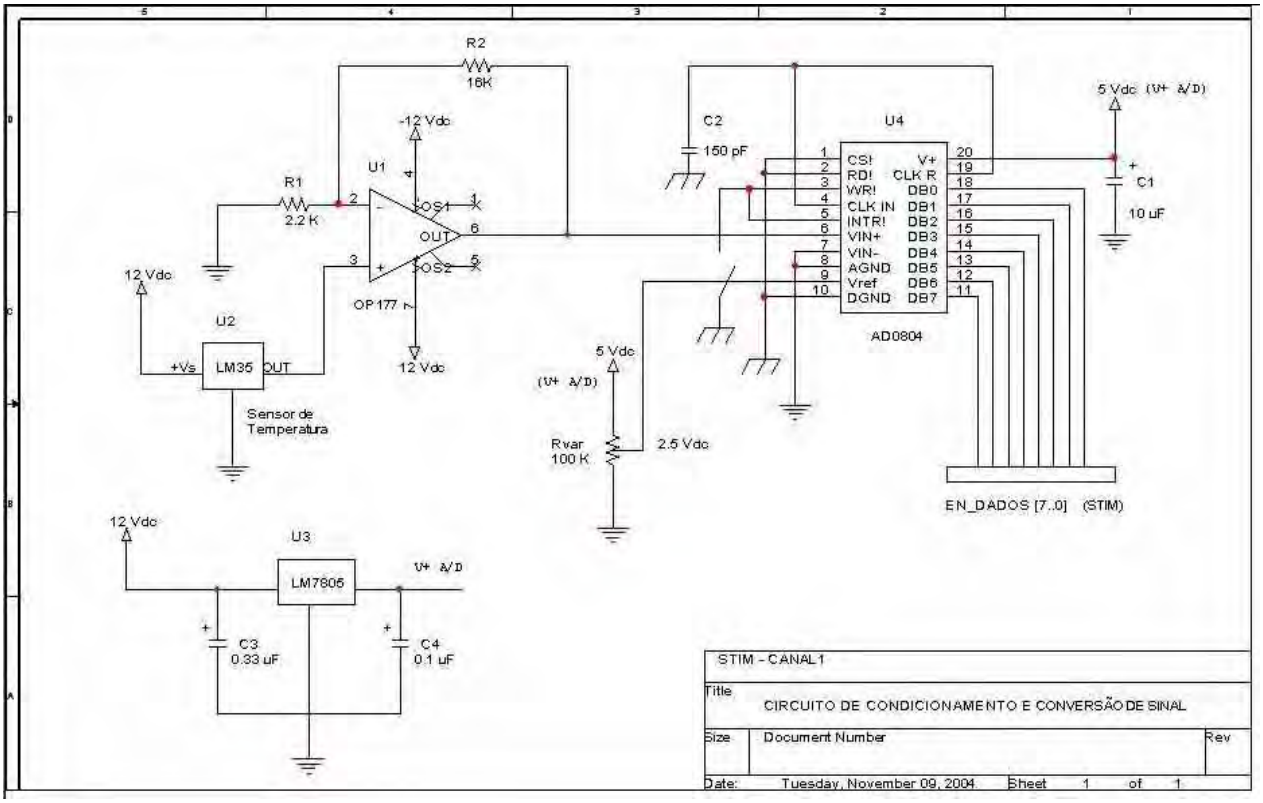


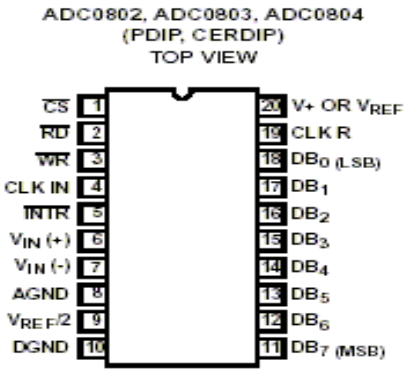
Figura F1. Diagrama esquemático dos circuitos de condicionamento e conversão de sinal, do canal 1.

Na Figura F2a, é apresentado o sensor de temperatura integrado, de uso geral, LM35 com as suas principais especificações e, na Figura F2b, o conversor A/D utilizado bem como suas características mais importantes.

a)
Fonte: *National Semiconductors. LM35 Precision Centigrade Temperature Sensors.*

LM35: Características Principais.

Sensor de temperatura linear: 10 mV/ºC
Exatidão: 0,5 ºC @ 25 ºC
Faixa de trabalho: -55ºC – 150 ºC
Alimentação: 4 – 30 V
Não linearidade: +/- ¼ ºC
Baixo custo, uso geral



b)
Fonte: *Intersil. ADC0802, ADC0803, ADC0804. 8-Bit, Microprocessor-Compatible, A/D Converters.*

AD0804: Características Principais.

8 *bits*, aproximações sucessivas
Alimentação (V+): 5 V
Tempo de conversão: <100 us
Entradas analógicas: 0 – 5 V
Tensão de referência (Vref): 2,5 V
Erro de quantificação: +/- 1 LSB @Vref: 2,5 V

Figura F2. a) Sensor integrado LM35, b) Conversor AD0804.

Na Figura F3 é mostrado o sistema sensor HMP45C, utilizado para realizar parte da comprovação experimental do modo *plug & play* e, na Figura F4, são mostrados os circuitos de condicionamento utilizados para o sensor de temperatura e para o sensor de umidade.

Sensor de temperatura PRT

1000 Ω, IEC 751 1/3 Classe B
Faixa de trabalho: -50ºC – 60ºC
Sinal de saída: 0,008 - 1 V
Alimentação: 7 – 35 V
Tempo de estabelecimento: 15 seg.

Sensor de umidade HUMICAP® 180

Faixa de trabalho: 0 – 100% RH
Sinal de saída: 0,008 – 1 V
Alimentação: 7 – 35 V
Tempo de estabelecimento: 15 seg.
Exatidão: +/- 2% (0 – 90% RH)

Figura F3. Sistema sensor HMP45C.

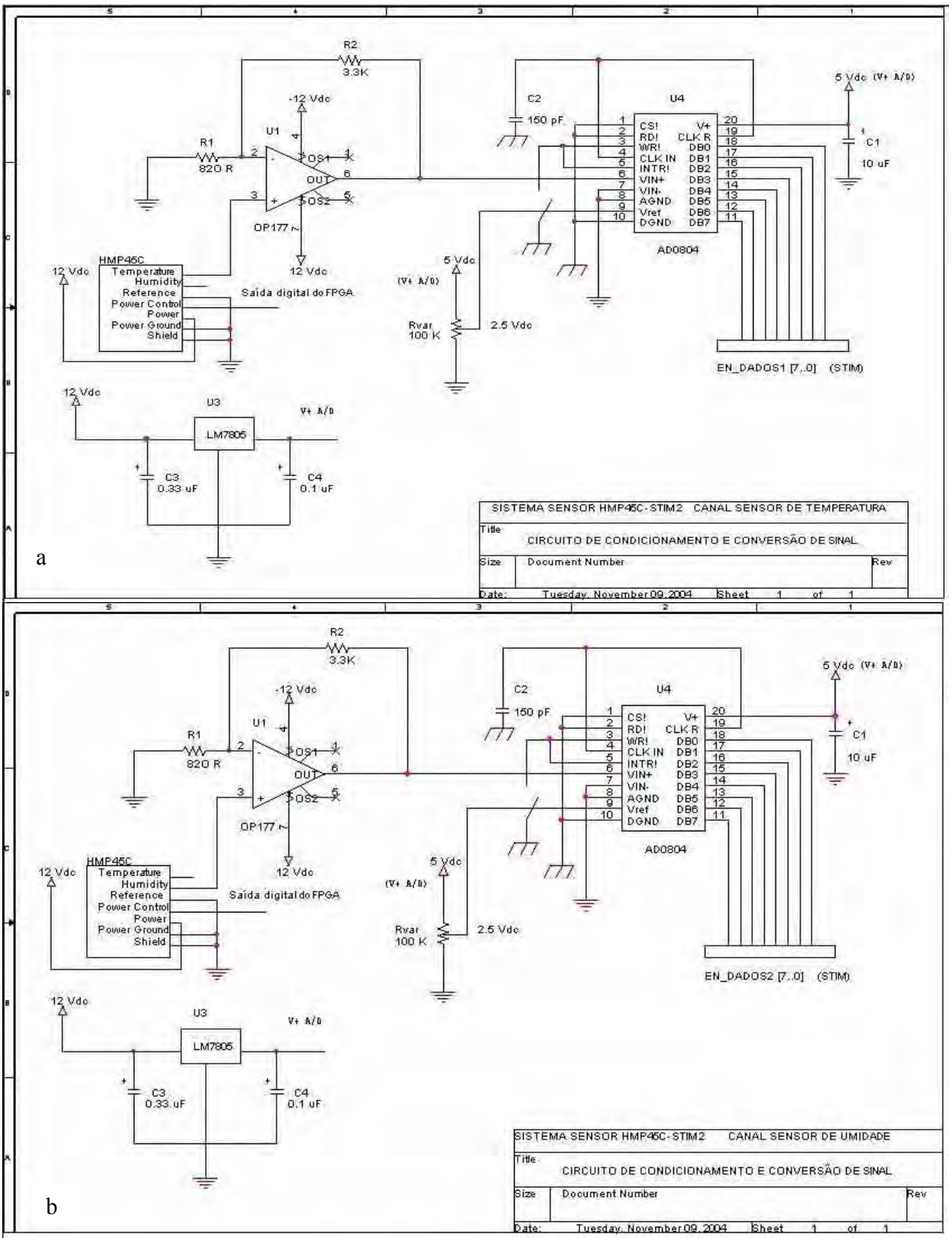


Figura F4. Diagramas esquemáticos dos circuitos de condicionamento e conversão de sinal para o STIM nº2: a) canal sensor de temperatura, b) canal sensor de umidade.