Ambiente para Desenvolvimento de Aplicações baseadas em Transdutores Inteligentes: Padrões IEEE 1451.2 e IEEE 1451.1

Alfranque Amaral da Silva

Dissertação de Mestrado submetida à Coordenadoria do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande - Campus de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Intrumentação Eletrônica

Antonio Marcus Nogueira Lima, Dr.
Orientador
Péricles Rezende Barros, PhD.
Orientador

Campina Grande, Paraíba, Brasil

Ambiente para Desenvolvimento de Aplicações baseadas em Transdutores Inteligentes: Padrões IEEE 1451.2 e IEEE 1451.1

Alfranque Amaral da Silva

Dissertação de Mestrado apresentada em Abril de 2005

Antonio Marcus Nogueira Lima, Dr.
Orientador
Péricles Rezende Barros, PhD.
Orientador

Campina Grande, Paraíba, Brasil, Abril de 2005

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

S586a Silva, Alfranque Amaral da
2005 Ambiente para Desenvolvimento de Aplicações baseadas em Transdutores Inteligentes: Padrões IEEE 1451.2 e IEEE 1451.1 / Alfranque Amaral da Silva. Campina Grande, 2005.
103f. : il.

Inclui bibliografia.
Dissertação (Mestrado em Engenharia Elétrica) – Universidade Federal de Campina Grande, Centro de Ciências e Tecnologia.
Orientadores: Antonio Marcus Nogueira Lima e Péricles Rezende Barros.

1 Sistemas Distribuídos Transdutores Inteligentes 2 Redes de Transdutores Inteligentes 3 Interfaces 4 Padrões IEEE 1451.1 e IEEE 1451.2 I Título

CDU 681.3.012+681.3.068

AMBIENTE PARA DESENVOLVIMENTO DE APLICAÇÕES BASEADAS EM TRANSDUTORES INTELIGENTES: PADRÕES IEEE 1451.2 E IEEE 1451.1

ALFRANQUE AMARAL DA SILVA

Dissertação Aprovada em 28.04.2005

ANTONIO MARCUS NOGUEIRA LIMA, Dr., UFCG
Orientador

PÉRICLES REZENDE BARROS, Ph.D., UFCG
Orientador

JOSE SÉRGIO DA ROCHA NETO, Dr., UFCG
Componente da Banca

EURICO BEZERRA DE SOUZA FILHO, D.Sc., UFCG
Componente da Banca

CAMPINA GRANDE - PB
Abril - 2005

Dedicatória

Este trabalho é dedicado especialmente a meu pai Alfredo Lobato da Silva e a minha mãe Francisca Corado do Amaral e Silva. Aos meus tios Joaquim Amaral Filho e Sílvia Corado do Amaral, grandes colaboradores e incentivadores, que estão sempre presentes nos momentos mais difíceis e a minha irmã Silvânia Amaral da Silva e meu sobrinho Pedro Augusto, minha grande fonte de alegria.

Agradecimentos

- À Deus que é Luz, Paz, Harmonia, a Verdadeira Força que nos faz e nos dá tudo que temos e somos;
- Aos meus pais Alfredo Lobato da Silva e Francisca Corado do Amaral e Silva, grandes timoneiros que sempre me guiaram pelos ditames do amor, carinho, otimismo, perseverança e honestidade, a quem sem dúvida alguma devo tudo que tenho e sou;
- Aos meus tios Joaquim Amaral e Sílvia Corado, grandes motivadores, colaboradores e incentivadores, que estiveram sempre presentes e participativos desde os tempos de cursinho;
- À minha irmã Silvânia Amaral e ao meu sobrinho Pedro Augusto, minha grande fonte de alegria;
- À minha avó Lavina Corado, fonte inesgotável de paz e serenidade;
- À minha tia Timotea Corado, grande educadora, pela alegria e confiança sempre presentes;
- À minha namorada Ceiça pelo apoio e compreensão dispensados;
- Aos meus grandes amigos Sérgio Murilo, José Alves, Orlei Oliveira, Arlindo de Sá, Rex Medeiros, Dênis Hipólito, Edmar José, Jaidilson Jó, Miguel Wanzeller, Félix Rodrigues, José Aleixo, Cicília Maia, Anderson Guedes, Tomas Victor, Luiz Felipe e tantos outros, pelo apoio, incentivo e sobretudo por estarem sempre presentes diante da menor solicitação de suas presenças;
- Aos meus orientadores Antônio Marcus Nogueira Lima e Péricles Rezende Barros pela paciência e pelas valiosas contribuições dadas;
- Aos companheiros do LIEC pela alegria e convivência fraterna;
- Aos professores e funcionários do Departamento de Engenharia Elétrica que de uma certa forma contribuíram para o andamento do meu trabalho;
- Ao CNPQ, pelo apoio financeiro.

Resumo

Nesta dissertação descreve-se a implementação de um ambiente de desenvolvimento, usado para desenvolvimento de transdutores inteligentes, segundo as especificações dos padrões IEEE 1451.2 e IEEE 1451.1. O padrão IEEE 1451.2 é designado como: Padrão para Interface de Transdutores Inteligentes para Sensores e Atuadores - Protocolos de Comunicação de Transdutor para Microcontrolador e Formatos dos TEDS (*Transducer Electronic Data Sheet*). O padrão IEEE 1451.1 é intitulado como: Padrão para Interface de Transdutores Inteligentes para Sensores e Atuadores - Informação do Modelo do NCAP (*Network Capable Application Processor*). Estes padrões foram implementados respectivamente para o microcontrolador ADuC832 e para a plataforma TINI (*Tiny InterNet Interface*). Como exemplo de utilização da plataforma foi desenvolvido um sistema de controle de temperatura que consiste de um STIM (*Smart Transducer Interface Module*) com um sensor de temperatura LM35 e um ventilador atuador. A leitura do sensor de temperatura e o controle do ventilador foram implementados para o ADuC832 utilizando uma placa de desenvolvimento com o ADuC832 (SAR *Eval Board* Rev A3) e a interface *web* foi codificada em Java e executada no TBM390 (TINI *Board Model* 390).

Abstract

In this dissertation is described the implementation of an development environment, used for intelligent transducers development, following the specifications of the IEEE1451.2 and IEEE 1451.1 standards. The IEEE1451.2 standard is designated as: Standard for a Smart Transducer Interface for Sensors and Actuators - Transducer to Micro-processor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats. The IEEE 1451.1 standard is designated as: Standard for a Smart Transducer Interface for Sensors and Actuators - Network Capable Application Processor (NCAP) Information Model. Those standards were implemented respectively for microconverter ADuC832 and for TINI (Tiny InterNet Interface) platform. As example of use of the platform was developed one system of temperature control that consists of a STIM (*Smart Transducer Interface Module*) with a sensor of temperature LM35 and a small fan actuator. The reading of the sensor of temperature and the control of the fan were implemented for ADuC832 using a evaluation board with the ADuC832 (SAR Eval Board Rev A3) and the interface web was codified in Java and executed in TBM390 (TINI Board Model 390).

Índice

Glossário

t_{dn}	Atraso válido de DIN para DCLK	[s]
f_{clk_max}	Máxima taxa de dados que o NCAP é capaz de suportar	[bits/s]
f_{clk_min}	Mínima taxa de dados que o NCAP é capaz de suportar	[bits/s]
f_{clk}	Taxa de dados selecionada pelo NCAP	[bits/s]
f_{max}	máxima taxa de dados suportado pelo STIM	[bits/s]
t_{ch}	Tempo alto do pulso de *clock*, linha DCLK	[s]
t_{cl}	Tempo baixo do pulso de *clock*, linha DCLK	[s]
t_{d}	Atraso válido de DCLK para DOUT	[s]
t_{edgen}	Tempo de subida ou descida: DCLK, DIN, NIOE, NTRIG	[μs]
t_{edges}	Tempo de subida ou descida: DOUT, NACK, NINT	[s]
t_{edgex}	Tempo de subida ou descida: DCLK	[s]
t_{grs}	Tempo de execução de leitura global	[s]
t_{gws}	Tempo de execução de escrita global	[s]
t_{hn}	Tempo de permanência inválido de DCLK para DOUT	[s]
t_{hold}	Tempo de *hold-off* do *byte*	[s]
t_{hs}	Tempo de execução de *handshake*	[s]
t_{h}	Tempo de permanência inválido de DCLK para DIN	[s]
t_{lat}	Tempo de detecção de latência do fim do quadro	[s]
t_{rs}	Tempo de execução de leitura de canal	[s]
t_{sp}	Período de amostragem de canal	[s]
t_{sun}	Tempo de execução DCLK para validar DOUT	[s]
t_{su}	DIN válido para tempo de execução DCLK	[s]
t_{u}	Tempo de atualização de canal	[s]
t_{wsp}	Pior caso do período de amostragem de canal	[s]

t_{ws}	Tempo de execução de escrita de canal	[s]
t_{wu}	Pior caso do tempo de atualização de canal	[s]
ADC	*Analog-to-Digital Converter*	
API	*Applications Programming Interface*	
APM	*Alternate Pulse Modulation*	
ASI	*Actuator-Sensor-Interface*	
ASIC	*Application-Specific Integrated Circuit*	
BEB	*Binary Exponential Backoff*	
CAN	*Controller Area Network*	
COMMON	Sinal comum ou terra	
CSMA/BA	*Carrier Sense Multiple Acess com Bitwise Arbitration*	
CSMA/CA	Acesso Múltiplo Sensível à Portadora com Mecanismo para Evitar Colisão	
CSMA/CD	Acesso Múltiplo Sensível à Portadora com Detecção de Colisão	
CSMA/DCR	Acesso Múltiplo Sensível à Portadora com Resolução de Colisão Determinística	
DAC	*Digital-to-Analog Converter*	
DCLK	DATA_CLOCK - Sinal de *clock*	
DHCP	*Dynamic Host Configuration Protocol*	
DI/O	*Digital Input/Output*	
DIN	DATA_IN - Transporta dados do NCAP para o STIM	
DNS	*Domain Name System*	
DOUT	DATA_OUT - Transporta dados do STIM para o NCAP	
DP	*Decentralized Periphery*	
FH-CDMA	*Frequency Hopping - Code-Division Multiple Access*	
FIP	*Factory Instrumentation Protocol*	
FMS	*Fieldbus Message Specification*	
FPGA	*Field Programmable Gate Array*	
FTP	*File Transport Protocol*	
gc	*garbage collection*	
HTTP	*HiperText Transfer Protocol*	

I/O	*Input/Output* - Entrada/Saída
ICMP	*Internet Control Message Protocol*
IEEE	*Institute of Electrical and Electronics Engineers*
JDK	*Java Development Kit*
JNI	*Java Native Interface*
JVM	*Java Virtual Machine*
LLC	*Link Logical Control*
MAC	*Medium Access Control*
MCU	*Microcontroller Unit*
MMT	*Mixed-Mode Transducer*
MN	Métodos Nativos
NACK	N_ACKNOWLEDGE - Reconhecimento de gatilhamento e transporte de dados
NCAP	*Network Capable Application Processor*
NINT	N_IO_INTERRUPT - Usado pelo STIM para requisitar serviço ao NCAP
NIOE	N_IO_ENABLE - Habilita transporte de dados
NIST	*National Institute of Standards and Technology*
NSDET	N_STIM_DETECT - Reconhecimento *plug-and-play* de um STIM
NTRIG	N_TRIGGER - Desempenha função de gatilhamento
OSI	*Open System Interconnection*
PCE	*Peripheral Chip Enable*
PDA	*Personal Digital Assistant*
POWER	Fonte de potência nominal de 5 V
PROFIBUS	*Process Fieldbuses*
SCRs	Sistemas de Controle em Redes
SI	Sistema Internacional de Unidades
SO	Sistema Operacional
SPI	*Serial Peripheral Interface*
STIM	*Smart Transducer Interface Module*
TBIM	*Transducer Bus Interface Modules*

TBM390	TINI - *Tiny InterNet Interface - Board Model* 390
TCP/IP	*Transmission Control Protocol / Internet Protocol*
TEDS	*Transducer Electronic Data Sheet*
TII	*Transducer Independent Interface*
TINI	*Tiny InterNet Interface*
UART	*Universal Asynchronous Receiver Transmitter*
WCM	*Wireless Communication Methods*
WSD	*Window Serial Donwloader*
XDCR	Transdutor

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Um transdutor inteligente pode ser definido como um sistema que é constituído de um transdutor analógico ou digital, um microprocessador e uma interface de comunicação. Esta funcionalidade tipicamente simplifica a integração do transdutor em aplicações no ambiente de rede, constituindo uma alternativa promissora para tratar adequadamente a complexidade inerente à inteligência distribuída nos nós da rede de transdutores de sistemas distribuídos.

O desenvolvimento de transdutores (sensores e atuadores) inteligentes, ocorreu como conseqüência do rápido avanço da micro-eletrônica nas últimas décadas. Motivados pela crescente integração de inteligência em produtos de entretenimento, brinquedos, utensílios domésticos, de escritórios e de uma forma geral, produtos desenvolvidos para utilização dos mais variados ramos da atividade humana. Estes produtos com algum tipo de inteligência embarcada[1] (WOLF, 2001), dependem cada vez mais de transdutores inteligentes para interagir com o mundo físico (JOHNSON; WOODS, 1998; LEE, 1999) no ambiente em que ele está instalado. Nesse contexto a utilização de transdutores inteligentes é justificada por vários aspectos, dentre eles: a capacidade de tomar decisões na fonte da informação (JOHNSON; WOODS, 1998); processamento local que converte dados em informação (JOHNSON, 1997), contribuído assim para reduzir a quantidade de dados que trafegarão na rede onde eles estão inseridos; auto-identificação; auto-teste; calibração adaptativa; facilidade de configuração; rejeição aprimorada de entradas espúrias (ruído) (JOHNSON; WOODS, 1998); facilidade de execução e uso (JOHNSON, 1997).

Simultaneamente ao rápido desenvolvimento da micro-eletrônica, iniciado nos anos setenta, que possibilitou o desenvolvimento de transdutores inteligentes os fabricantes dos mesmos passaram a integrar interfaces de redes a estes transdutores, objetivando integrá-los em um ambiente de rede. O uso de redes era eminentemente motivado pelas seguintes

[1] Um sistema embarcado é definido como qualquer dispositivo que inclui um programa de computador, mas que por si só não é entendido como um computador de propósito geral.

razões: redução do custo de cabeamento, modularização dos sistemas e flexibilidade na configuração do sistema (JOHNSON, 1997). Diversos tipos de redes e protocolos de redes foram desenvolvidos para suprir uma demanda crescente e diversificada de aplicações. Entretanto, o fato dos transdutores serem desenvolvidos para redes específicas, como a rede ASI (*Actuator-Sensor-Interface*) e PROFIBUS, isso inviabilizava a integração dos mesmos nos diversos ambientes de redes existentes e incompatíveis uns com os outros. Este fato, obviamente, é um complicador na disseminação e utilização das redes de transdutores pelos diversos seguimentos industriais e da atividade humana de uma maneira geral. Além disso, a não padronização das interfaces de rede limitava o desenvolvimento de transdutores às grandes corporações que investiam somas altíssimas no desenvolvimento de seus próprios transdutores e protocolos de redes. Verdadeiros monopólios eram criados, restringindo grandes fatias do mercado a um número limitado de fabricantes e conseqüentemente impedindo que pequenos fabricantes desenvolvessem seus produtos e competissem com eles no mercado. A conseqüência imediata deste fato são os altos preços dos transdutores e das redes. Por outro lado, isto forçava os usuários de redes e transdutores a utilizarem um tipo específico de rede ou transdutor. Assim, uma vez feita a escolha por este ou aquele transdutor e sua respectiva rede, o usuário ficava "refém" destas tecnologias dificultando, caso necessário, a migração ou expansão para um outro tipo de rede ou transdutor, sob pena de ter que re-investir novas e grandes somas na aquisição de uma nova rede e de novos transdutores.

Assim diversos tipos de transdutores, redes e protocolos de redes foram desenvolvidos para suprir uma demanda crescente e diversificada de aplicações. Na ausência de uma padronização a migração de aplicações baseadas em transdutores analógicos ou digitais para um ambiente de redes de transdutores inteligentes exigirá, a cada vez e a cada caso a solução do problema de escolha do melhor tipo de rede e, conseqüentemente, o projeto da interface de comunicação apropriada; a adoção dos padrões desenvolvidos pelo grupo de trabalho IEEE 1451 (NIST, 2005) é uma das alternativas para simplificar a migração de um sistema de instrumentação convencional para um sistema de instrumentação em rede. Neste caso, para um ambiente de rede com fios é necessário projetar o NCAP (*Network Capable Application Processor*), padrão IEEE 1451.1, e o STIM (*Smart Transducer Interface Module*), padrão IEEE 1451.2, que ainda não são disponíveis comercialmente na forma de circuitos integrados.

Um outro ponto relevante a ser considerado nas aplicações baseadas em redes de transdutores inteligentes é o aspecto heterogêneo das etapas de desenvolvimento. No desenvolvimento de uma aplicação cujo objetivo seja a visualização dos dados gerados de um sensor inteligente é necessário escrever um programa aplicativo de rede que de um lado implemente a comunicação com o transdutor inteligente e do outro a comunicação

com a rede.

A padronização de transdutores e das *interfaces* de redes permitiria a atenuação dos gargalos de interconexões e o rápido desenvolvimento de sensores e atuadores inteligentes para uso em redes, além de reduzir o custo de desenvolvimento de sensores inteligentes e disseminar o seu uso. O surgimento de dispositivos inteligentes e redes de controle, tornaria disponível uma grande variedade de produtos, para usuários escolhê-los, baseado em seus méritos. Se esta tendência continuar, a médio prazo, o custo total do sistema será reduzido e o domínio de aplicações de controle e medições distribuídos será ampliado (SCHNEEMAN, 1999).

1.1 Família de padrões IEEE 1451

O cenário de não padronização das interfaces de redes e transdutores inteligentes levou o IEEE (*Institute of Electrical and Electronics Engineers*) propor a padronização das interfaces de redes, a nível de *hardware* e *software*, para possibilitar interligar transdutores inteligentes a qualquer tipo de rede. A partir de 1993 o IEEE e o NIST (*National Institute of Standards and Technology*) promoveram uma série de *workshops* para discutir as interfaces para sensores inteligentes e a possibilidade de desenvolvimento de um padrão de interfaces que simplificasse a conectividade de transdutores inteligentes em rede. O resultado desses *workshops* foi a criação de 6 grupos de trabalho intitulados IEEE 1451 (NIST, 2005). Na Figura 1.1 são ilustradas as denominações e as relações entre os 6 padrões, objeto dos grupos de trabalho, doravante descritos nas subseções seguintes.

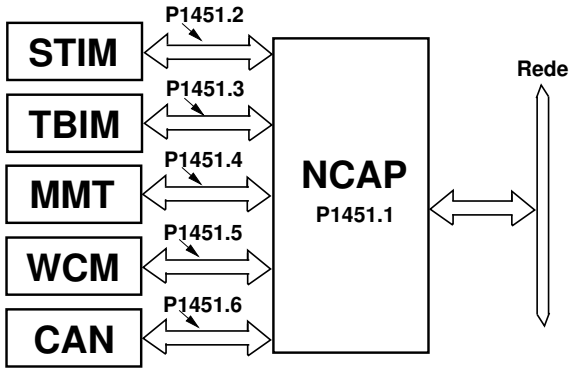

Figura 1.1: Relação entre os padrões da família IEEE 1451.

1.1.1 O grupo de trabalho P1451.1

O grupo de trabalho P1451.1 teve como objetivo definir a informação do modelo do NCAP (*Network Capable Application Processor*), consistindo de um modelo de objeto

comum para transdutores inteligentes juntamente com a especificação de interfaces para os componentes do modelo. Este projeto deu origem ao padrão IEEE 1451.1, publicado em 1999 (IEEE, 1999).

1.1.2 O grupo de trabalho P1451.2

O grupo de trabalho P1451.2 teve como meta definir os TEDS (*Transducer Electronic Data Sheet*), o STIM (*Smart Transducer Interface Module*) e a interface digital, incluindo alocação de pinos de conector e protocolo de comunicação entre o STIM e o NCAP. Este projeto deu origem ao padrão IEEE 1451.2, publicado em 1997 (IEEE, 1997).

1.1.3 O grupo de trabalho P1451.3

O grupo de trabalho P1451.3 teve como meta definir uma interface de comunicação digital para sistemas distribuídos multiponto, onde uma única linha de transmissão é usada para alimentar todos os transdutores e fornecer a comunicação entre o controlador de barramento, contido no NCAP, e os TBIMs (*Transducer Bus Interface Modules*). Este projeto deu origem ao padrão IEEE 1451.3, publicado em 2004 (IEEE, 2004a).

1.1.4 O grupo de trabalho P1451.4

O grupo de trabalho P1451.4 teve como objetivo definir um protocolo de comunicação para transdutores inteligentes operando no modo misto (MMT - *Mixed-Mode Transducer* (LEE; CHEN, 1998)), isto é, com modos de operação analógica ou digital. Este projeto deu origem ao padrão IEEE 1451.4, publicado em 2004 (IEEE, 2004b).

1.1.5 O grupo de trabalho P1451.5

O grupo de trabalho P1451.5, ainda em atividade, está incumbido de estabelecer um padrão para modos de comunicação sem fio (WCM - *Wireless Communication Methods*) e o formato dos dados para os transdutores. Este padrão definirá os TEDS de acordo com o conceito IEEE 1451 e os protocolos para acesso dos TEDS e dos dados de transdutores (NIST, 2005).

1.1.6 O grupo de trabalho P1451.6

Ao grupo de trabalho P1451.6, ainda em atividade, está encarregado de estabelecer um padrão para operação de um transdutor e um controlador em malha fechada em um ambiente de rede em cascata com múltiplos controladores em cada nível. A camada de transporte da rede é um barramento serial CAN (*Controller Area Network*) . O padrão

IEEE P1451.6 define de forma básica e segura uma camada física aberta que combina os padrões IEEE 1451 e tecnologias para redes CAN abertas (IEEE, 2005).

1.2 Redes no ambiente de controle

As redes de comunicação passaram a fazer parte do cenário dos sistemas de controle digital nos anos setenta visando atender à demanda da indústria automotiva. Exemplos de sistemas de controle que utilizam redes são encontrados na automação industrial, em sistemas de veículos inteligentes, em aeronaves (CHOW; TIPSUWAN, 2001). Um diagrama ilustrativo do tipo de ambiente onde são utilizados transdutores inteligentes é apresentado na Figura 1.2. Neste ambiente, os módulos transdutores inteligentes são inseridos em processos de supervisão, medição e/ou controle industrial, residencial e predial, como ilustrado pela região delimitada pela linha tracejada. Os dados oriundos ou destinados ao módulo transdutor inteligente podem ser visualizados ou definidos, por meio de uma interface escrita em Java e executada de dentro de uma página *web*, disponibilizada para acesso via rede *intranet/internet*.

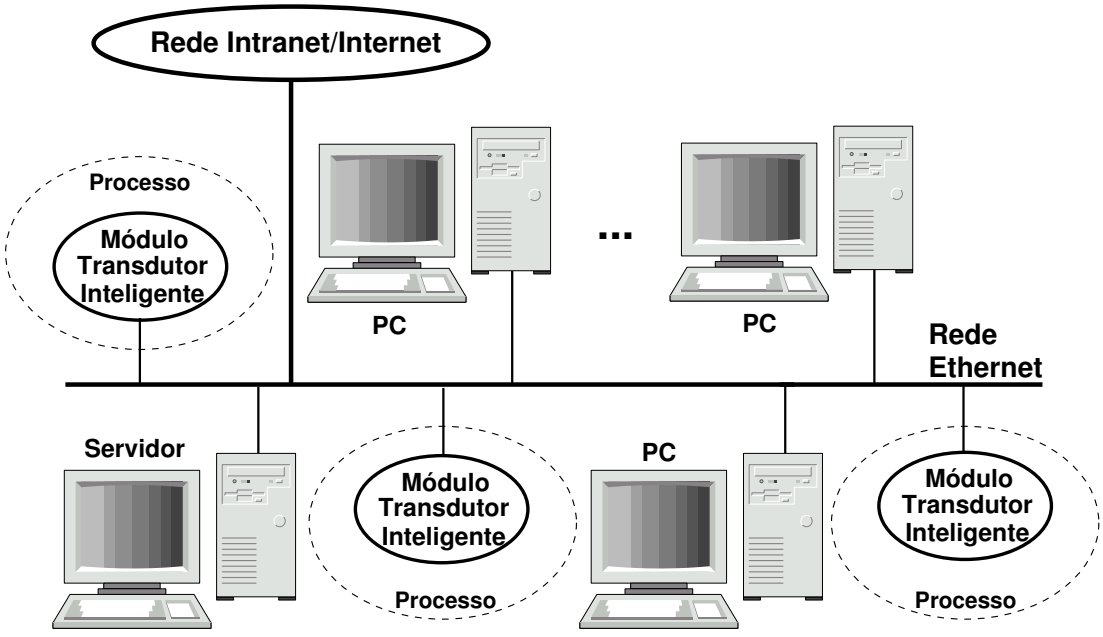

Figura 1.2: Rede Ethernet, com transdutores inteligentes inseridos em processos de medição e/ou controle.

1.3 Objetivos e organização do trabalho

O objetivo deste trabalho é implementar um ambiente para desenvolvimento de aplicações baseadas em transdutores inteligentes, segundo os padrões IEEE 1451.2 (IEEE, 1997) e

IEEE 1451.1 (IEEE, 1999). O padrão IEEE 1451.2 foi implementado para o microcontrolador ADuC832 (ANALOG DEVICES, 2002a) e o padrão IEEE 1451.1 para o TINI (*Tiny InterNet Interface*) modelo 390 (DALLAS, 2001). A descrição dos dispositivos utilizados para a construção do ambiente de desenvolvimento, usado para a implementação de transdutores inteligentes e as etapas de desenvolvimento dos mesmos, serão descritos ao longo desta dissertação, a qual está estruturada como a seguir. No capítulo 2 é apresentado o módulo transdutor inteligente segundo a especificação dos padrões IEEE 1451.1 e IEEE 1451.2. No capítulo 3 é apresentado o *hardware* usado na implementação do STIM e do NCAP, a *interface* SPI (*Serial Peripheral Interface*) e os conceitos da JNI (*Java Native Interface*). No capítulo 4 são apresentados o ambiente de desenvolvimento, os resultados experimentais obtidos e a estrutura do aplicativo do NCAP com a interface gráfica. No capítulo 5 são apresentadas as conclusões e perspectivas de trabalhos futuros. Finalmente no apêndice A são apresentadas algumas redes de comunicação e de campo usadas no ambiente de controle e supervisão.

Capítulo 2

Módulo transdutor inteligente

2.1 Introdução

O foco deste capítulo é definir um conjunto de regras, protocolos e especificações físicas e funcionais para estabelecer os requisitos do módulo transdutor inteligente, segundo os padrões IEEE 1451.1 (IEEE, 1999) e IEEE 1451.2 (IEEE, 1997). Neste sentido, este capítulo tem uma relevância vital para o entendimento desta dissertação.

2.2 O módulo transdutor inteligente

O módulo transdutor inteligente mostrado na Figura 1.2 é especificado pelo padrão IEEE 1451.2 (IEEE, 1997) e é da forma ilustrado na Figura 2.1. A interface digital com o transdutor inteligente é feita através da interface TII (*Transducer Independent Interface*). Nesta interface, o transporte de dados do STIM (*Smart Transducer Interface Module*) para o NCAP (*Network Capable Application Processor*) e vice-versa é feito utilizando a interface SPI (*Serial Peripheral Interface*) (ESTL, 2002).

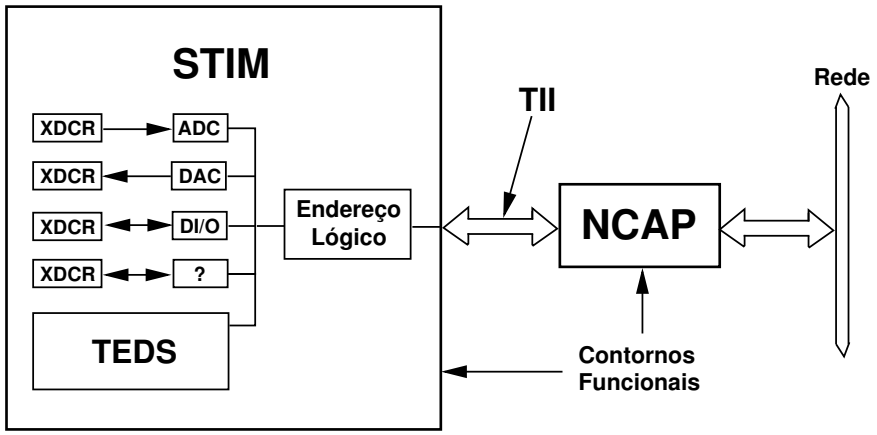

Figura 2.1: Contexto para a especificação da interface do módulo transdutor inteligente.

Na Figura 2.1 o bloco XDCR representa um transdutor. Os blocos ADC (*Analog-to-Digital Converter*), DAC (*Digital-to-Analog Converter*), DI/O (*Digital Input/Output*) e ? forma de I/O (*Input/Output* - Entrada/Saída) diferente das anteriormente citadas, representam formas de entradas e saídas de dados. O bloco TEDS (*Transducer Electronic Data Sheet*) diz respeito aos TEDS, onde são armazenados os dados eletrônicos portando as especificações e toda descrição do XDCR. O bloco endereço lógico, especifica toda a lógica de canais dos XDCRs. O bloco NCAP, também chamado nó de rede, integra o STIM em uma rede qualquer. Os contornos funcionais dizem respeito à distribuição físico-espacial do STIM e do NCAP do ponto de vista da conexão dos mesmos, via interface TII implementada no STIM e no NCAP. Geralmente contornos funcionais são implementados em chips diferentes. Um exemplo de STIM e NCAP com contornos funcionais implementado em um único chip, ou seja, o NCAP e o STIM implementados em um único chip, é apresentado em (WALL; EKPRUKE, 2003). Neste caso a interface TII é implementada em *software*. A rede para o NCAP desenvolvido e apresentado neste trabalho é a rede *Ethernet*. Segue-se uma descrição detalhada das especificações funcionais de um transdutor inteligente segundo o padrão IEEE 1451.2 e de cada bloco da Figura 2.1.

2.3 Formas de I/O

O bloco **?**, mostrado na Figura2.1, representa tipos de I/O diferentes dos blocos de I/O convencionais, como: ADC, DAC, DI/O. Já o bloco XDCR representa um transdutor (sensor ou atuador) qualquer que interfaceado por um microcontrolador executando as funções de endereçamento, escrita, leitura, gatilhamento, juntamente com os TEDS, ambos definidos no padrão IEEE 1451.2 formam um STIM. Um exemplo de XDCR é um sensor de temperatura LM35. Outro exemplo é um atuador como um motor ou uma válvula.

2.4 Especificações funcionais de um STIM

Nesta seção será especificado os tipos de canais de transdutores que podem ser implementados em um STIM e as funções requisitadas do STIM para permitir o funcionamento dos canais conjuntamente. Também são definidas funções de: endereçamento, transporte de dados, gatilhamento, controle e *status*.

2.4.1 Tipos de canais de transdutores

O padrão IEEE 1451.2 especifica o comportamento de 6 tipos de canais. Um sétimo tipo de canal, já previsto no padrão, é identificado para permitir extensões do comportamento

do STIM, além das aqui especificadas. Os 7 tipos de canais são os seguintes:

1. **Sensor** - mede algum parâmetro físico sob demanda e retorna dados digitais que representam aquele parâmetro. Novos dados devem ser amostrados apenas depois de um evento de gatilhamento. O conjunto de dados disponível para leitura deve ser o conjunto de dados adquirido como resultado do mais recente evento de gatilhamento;

2. **Atuador** - um atuador causa a ocorrência de uma ação física ou virtual que deve está relacionada ao conjunto de dados enviados ao atuador. O estado de um atuador muda para se adaptar ao mais recente conjunto de dados recebidos quando ocorre um evento de gatilhamento;

3. **Sensor com** *buffer* - um sensor com *buffer* difere de um sensor simples apenas por possuir um único nível de dados no *buffer* sobre a saída do canal. Todas os outros aspectos de funcionamento são idênticos ao do sensor simples.

4. **Sensor de seqüência de dados** - um sensor de seqüência de dados adquire dados continuamente com tempos de amostragem sob controle do STIM. O gatilhamento seleciona um conjunto de dados desse fluxo (*stream*) de medições contínuas e os deixa disponíveis para o NCAP efetuar uma leitura. O conjunto de dados selecionado deve ser adquirido imediatamente depois do gatilhamento. O processo de aquisição de dados no STIM deve ser habilitado ou desabilitado por meio de um comando de controle. A temporização para aquisição de um conjunto de dados não precisa ser periódica.

5. **Sensor de seqüência de dados com** *buffer* - um sensor de seqüência de dados com *buffer* adquire dados continuamente com tempos de amostragem sob controle do STIM. O gatilhamento seleciona um conjunto de dados desse fluxo (*stream*) de medições contínuas e os deixa disponíveis para o NCAP efetuar uma leitura. O conjunto de dados selecionado deve ser adquirido imediatamente antes do gatilhamento. O processo de aquisição de dados no STIM deve ser habilitado ou desabilitado por meio de um comando de controle. A temporização para aquisição do conjunto de dados não precisa ser periódica.

6. **Sensor de seqüência de eventos** - um sensor de seqüência de eventos produz um sinal se um evento específico ocorrer. O sinal deve ser o mesmo sinal usado pelos sensores e atuadores para reconhecer eventos de gatilhamento. O evento pode ser uma transição de um sinal digital ou um cruzamento de limite de referência de um sinal analógico. O sinal pode ser configurado para um evento de transição de alto para baixo ou de baixo para alto ou ambos. Leitura e/ou escrita sobre um

canal de um sensor de seqüência de eventos deve ocorrer apenas para informação do estado e configuração. O tempo do evento é guiado pelo sinal de reconhecimento de gatilhamento. Outros canais de sensor e atuador podem ser associados com um sensor de seqüência de eventos para permitir mudanças de limites de referências ou histereses analógicas, ou ler um valor analógico sendo enviado. Esta associação é comunicada ao NCAP através do grupo de informações do Meta-TEDS;

7. **Transdutor genérico** - esta categoria de canal permite a presença de canais que se comportam de forma diferente dos tipos definidos anteriormente. Estes tipos devem implementar as mesmas funções requisitadas de todos os canais, mas o padrão não especifica o comportamento com respeito ao gatilhamento e escrita ou leitura de dados de tais canais.

2.4.2 Modo de endereçamento

O endereçamento é usado em conjunto com a interface de transporte de dados. O endereço completo especifica se dados estão sendo lidos ou escritos e para qual função. O endereço completo deve ser de 2 bytes e é estruturado como na Figura 2.2. O endereço completo corresponde ao bloco endereço lógico da Figura 2.1

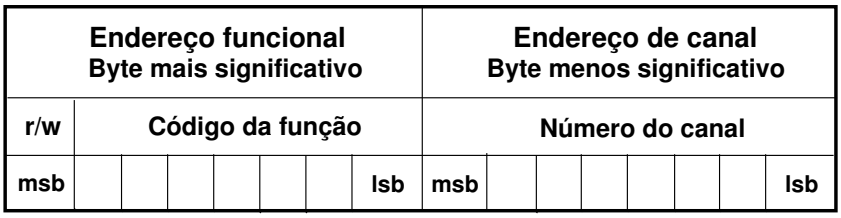

Figura 2.2: Leiaute do endereço completo.

O endereço funcional e de canal são endereços lógicos. Por conveniência, o *byte* mais significativo é chamado endereço funcional e o menos significativo endereço de canal. A função do endereço funcional está subordinada ao endereço de canal, mas é transmitido primeiro para tornar a implementação mais fácil.

A cada transdutor no STIM deve ser designado um endereço de canal. Um STIM pode ter até 255 canais. O número de canais implementados é determinado lendo o Meta-TEDS. O canal zero tem significado especial e é chamado de CHANNEL_ZERO. Quando o canal zero é usado, a função deve se referir ao STIM como um todo ou a todos os canais coletivamente. O msb do endereço funcional é usado para especificar a direção da comunicação sobre a interface, isto é:

- 0 - escrita para o STIM;

- 1 - leitura do STIM;

Os demais *bits* remanescentes do endereço funcional denotam a função selecionada.

2.4.3 Interface para transporte de dados

O transporte de dados deve acontecer em quadros de números inteiros de *bytes*. O *byte* mais significativo deve ser enviado primeiro. A interface física deve transportar cada *byte* na forma de *bit* serial, sendo o *bit* mais significativo de cada byte transmitido primeiro.

A função de transporte de dados deve interagir com a função de gatilhamento e deve está inativa antes do gatilhamento ser declarado. O processo de transferência de quadros deve começar com o NCAP enviando o endereço para o STIM. O endereço completo especifica se o dado deve ser escrito ou lido do STIM e qual canal e função estão envolvidos.

Dados do transdutor

O transporte de dados deve ser freqüentemente usado para ler dados de sensores, sensores com *buffer*, sensores de seqüência de dados e sensores de seqüência de dados com *buffer*, e para escrever dados para atuadores e canais de seqüência de eventos.

Não tem efeito escrever dados para sensores, sensores com *buffer*, sensores de seqüência de dados e sensores de seqüência de dados com *buffer*. Ler dados de qualquer tipo de sensor, sem antes gatilhá-lo, deve retornar os últimos dados da última leitura.

Ler dados de qualquer sensor depois de abortar um ciclo de gatilhamento pode produzir resultados imprevisíveis. Ler dados de um sensor com *buffer* ou sensor de seqüência de dados com *buffer*, depois da inicialização ou depois de abortado o cíclo de gatilhamento, pode produzir resultados imprevisíveis, até mesmo depois do segundo gatilhamento ser enviado. Ler dado de um atuador deve retornar o último dado escrito para ele. Ler dados de um atuador depois da inicialização, mas antes de escrever dados para ele, deve retornar o estado inicial do atuador.

As funções do canal zero de escrita e/ou leitura global de dados do transdutor resulta na escrita ou leitura da estrutura de dados de todos os canais implementados concatenadamente, em ordem, iniciando pelo canal 1. Isso permite acessar todos os transdutores sem o *overhead* associado ao endereçamento de cada canal individual. Não existem delimitadores entre estruturas de dados de canal. O dado para cada canal deve está de acordo com o modelo de dados especificado em cada TEDS de canal. O comportamento de cada canal individual deve adequar-se ao comportamento especificado para tais leituras e escritas de canais individuais.

Escrever um endereço de canal para um canal não implementado ou um endereço funcional inválido de um canal implementado não deve mudar ou corromper o estado de

qualquer função válida de qualquer canal implementado. O STIM deve responder a qualquer tentativa de endereçar canais não implementados setando o *bit* de comando inválido do STIM no registrador de *status* padrão. Veja seção 2.4.6 para verificar a descrição desse *bit*.

2.4.4 Função de gatilhamento

A função de gatilhamento proporciona meios para um NCAP enviar ao STIM um comando para solicitar uma ação (gatilhar o sinal), e para o STIM sinalizar quando a ação ocorreu (reconhecimento de gatilhamento). Cada tipo de canal de transdutor difere do outro principalmente na maneira que ele responde ao gatilhamento. O diagrama de gatilhamento do sistema, do ponto de vista do STIM, é ilustrado na Figura 2.3 e é detalhado através do protocolo de gatilhamento apresentado na seção 2.6.1.

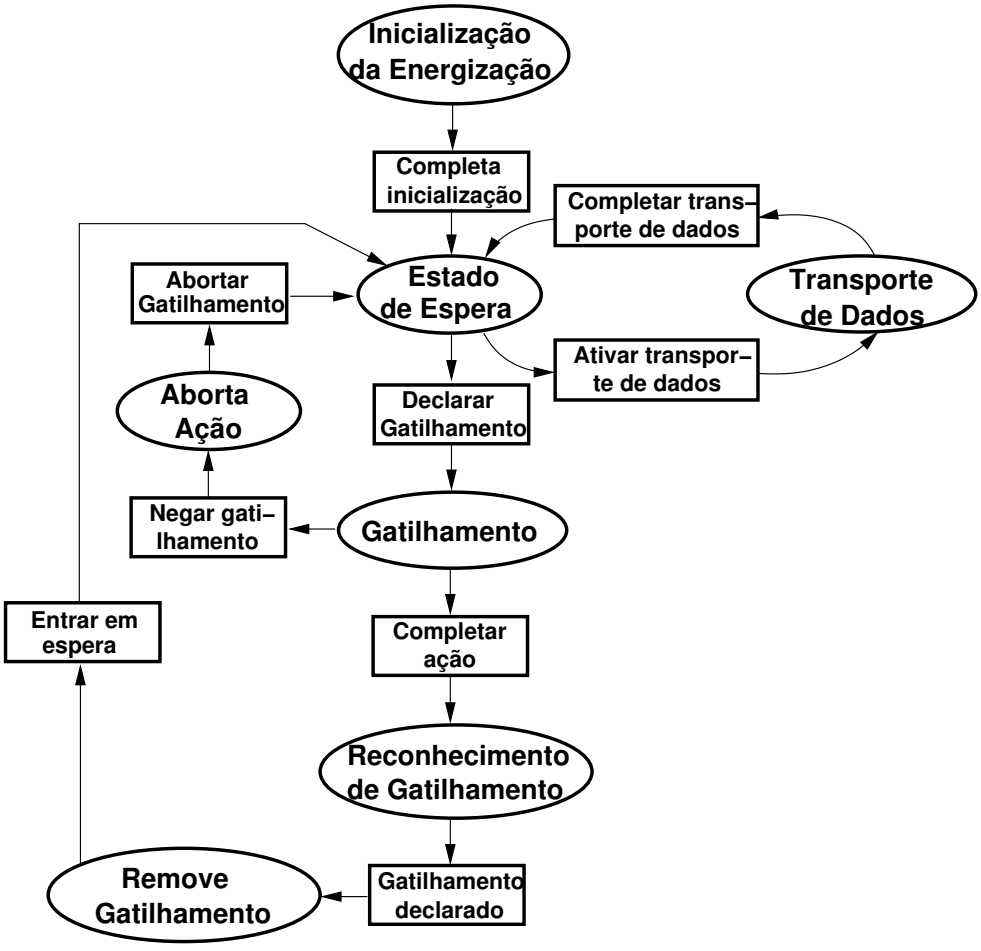

Figura 2.3: Resposta genérica do STIM para um gatilhamento

A ação iniciada por um gatilhamento normal se diferencia a partir do tipo de canal dependente ou processo concorrente. O gatilhamento deve ser aplicado apenas a um único canal ou a todos os canais (gatilhamento global). O canal para o qual o gatilhamento é

aplicado é selecionado pelo endereço do canal gatilhado.

Na Tabela 2.1 são mostrados os estados relacionados ao gatilhamento.

Tabela 2.1: Estados do STIM relacionados ao gatilhamento

Estado	**Definição**
Início da alimentação	O STIM está desempenhando a inicialização.
Quiescente	O STIM está esperando para gatilhar ou para ativar o transporte de dados.
Gatilhado	O STIM recebeu um gatilhamento e o canal gatilhado está desempenhando ações apropriadas.
Aborta ação	O NCAP retirou o gatilhamento prematuramente para abortar a ação no STIM. O transdutor resultante tem dado ou estado do atuador indefinido. O estado da máquina precede diretamente ao estado quiescente.
Reconhecimento de gatilhamento	O STIM reconheceu o gatilhamento e espera o NCAP negá-lo.
Remove gatilhamento	O NCAP negou o gatilhamento. O STIM remove o reconhecimento de gatilhamento. O estado de máquina precede diretamente o estado quiescente.
Transporte de dados	O STIM está lendo ou escrevedo dados, *status*, controle ou campo de dados do TEDS. O gatilhamento deve ficar inativo (negado) até ser completado o transporte de dados.

Quando canal gatilhado for o canal zero, todos os canais devem responder a um sinal de gatilhamento. Devido a possíveis permutações de tipos de transdutores não é possível especificar, no caso geral, a resposta ao gatilhamento global.

2.4.5 Funções de controle

A função de controle permite que comandos sejam enviados ao STIM como um todo ou para cada canal e afeta o estado ou operação do canal. Ele deve ser acessado pelo NCAP através da escrita do endereço funcional através da interface física, escrevendo o comando de controle para escrita de canal, para um canal específico ou comando de controle para escrita global, para o canal zero. Os comandos de controle devem ter 2 *bytes* e estão listados na Tabela 2.2. Os comandos de controle 0 e 1 devem ser implementados por todo STIM e para todos os canais. O STIM deve responder a todos os comandos não implementados setando o *bit* de comando inválido no registrador de *status* padrão.

Tabela 2.2: Comandos padrões de controle

Valor	**Definição CHANNEL_ZERO**	**Definição canal individual**
0	Não operante	Não operante
1	Reinicializa STIM	Reinicializa canal
2	Inicializa auto-teste do STIM	Inicializa auto-teste de canal
3	Calibra todos os canais	Calibra canal
4	Zera todos os canais	Zera canal
5	Habilita todos os sensores de seqüência de eventos	Habilita sensor de seqüência de evento
6	Desabilita todos os sensores de seqüência de eventos	Desabilita sensor de seqüência de evento
7	Seta todos os sensores de seqüência de eventos para o modo de configuração	Seta sensor de seqüência de evento para o modo de configuração
8	Reservado	Reservado
9	Habilita todos os sensores de seqüência de dados ou de seqüência de dados com *buffer*	Habilita sensores de seqüência de dados ou de seqüência de dados com *buffer*
10	Desabilita todos os sensores de seqüência de dados ou de seqüência de dados com *buffer*	Desabilita sensores de seqüência de dados ou de seqüência de dados com *buffer*
11-255	Reservado	Reservado
256-65 535	Aberto para indústria	Aberto para indústria

2.4.6 Status

A função de *status* permite ao NCAP determinar o estado global do STIM ou de canais individuais. Ele é implementado por meio de um registrador de *status* padrão e auxiliar. Cada *bit* no registrador de *status* especifica a presença ou ausência de uma condição particular. A presença de uma condição deve ser representada pelo *bit* 1 na posição apropriada.

A função de *status* também é usada em conjunto com as máscaras de interrupção e as interrupções para indicar que o STIM está requisitando serviço e para que propósito. Os *bits* de *status*, tidos como reservados e não implementados, devem ser reportados como 0 quando lidos.

Status padrão

O registrador de *status* padrão deve ser acessado pela leitura do endereço funcional, a partir da leitura do *status* padrão do canal, para o canal em questão ou lendo o endereço funcional de *stutus* do padrão global para o canal zero. O *status* retornado deve ser de 2 *bytes*. Os *bits* de *status* do padrão global devem permitir implementar:

- *bit* de requisição de serviço global;
- *bit* de reconhecimento de gatilhamento do STIM;
- *bit* de *reset* do STIM;
- *bit* de comando inválido do STIM;
- *bits* disponíveis para *status* auxiliar de todos os canais do STIM OR;
- *bit* operacional do STIM.

Os *bits* de *status* devem permitir implementar os seguintes canais padrões:

- *bit* de requisição de serviço de canal;
- *bit* de reconhecimento de gatilhamento do canal;
- *bit* de *reset* de canal;
- *bit* disponível para *status* auxiliar de canal;
- *bit* operacional de canal.

Status auxiliar

O registrador de *status* auxiliar deve ser acessado pela leitura do endereço funcional, a partir da leitura do *status* auxiliar do canal para o canal em questão e lendo o endereço funcional de *status* auxiliar global para o canal zero. O *status* retornado deve ter 2 *bits*. Todos os *bits* de *status* auxiliar são opcionais.

2.4.7 Máscara de interrupção

O STIM deve conter registradores de máscaras de interrupção padrão e auxiliar, ambos de 2 *bytes*. Ao escrever 1 em qualquer posição do registrador de máscara de interrupção habilitará o *bit* de requisição de serviço para ser setado quando o correspondente *bit* no registrador de *status* é setado. Uma requisição de serviço é gerada por um circuito

combinacional como mostrado na Figura 2.4 para detalhes. Quando o *bit* de requisição de serviço é setado, uma interrupção será gerada.

A expressão lógica que rege o circuito combinacional da Figura 2.4 é dada por:

$$rs = (mip\ \ AND\ \ rsp)\ \ OR\ \ (mia\ \ AND\ \ rsa)$$

onde: rs é a requisição de serviço, mip é a máscara de interrupção padrão, mia é a máscara de interrupção auxiliar, rsp é o registrador de *status* padrão e rsa é o registrador de *status* auxiliar.

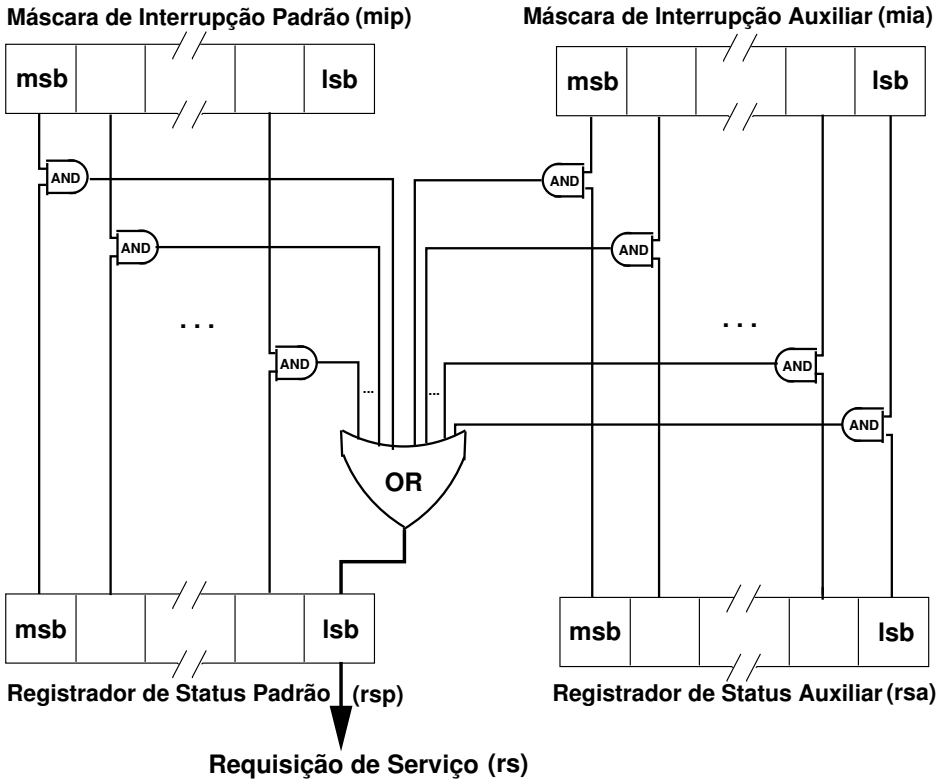

Figura 2.4: Máscara de interrupção de canal ou global.

As posições dos *bits* no registrador de máscara de interrupção padrão correspondem um a um com a posição dos *bits* no registrador de *status* padrão. O valor *default* para todos os *bits* no registrador de máscara de interrupção padrão é 1. Isto é, todos os *bits* de *status* padrão podem gerar interrupções. O valor *default* para todos os *bits* do registrador de máscara de interrupção auxiliar é 0. Isto é, *bits* de *status* auxiliar não podem gerar interrupções. Um comando de *reset* do STIM não deve afetar o valor desses registradores.

2.4.8 Interrupções

Um sinal digital adicional separado na interface física deve ser fornecido ao STIM para requisitar serviços ao NCAP. O sinal de interrupção deve ser declarado se o *bit* de requi-

sição de serviço é setado no registrador de *status* padrão global ou em qualquer registrador de *status* padrão de canal (veja Figura 2.5).

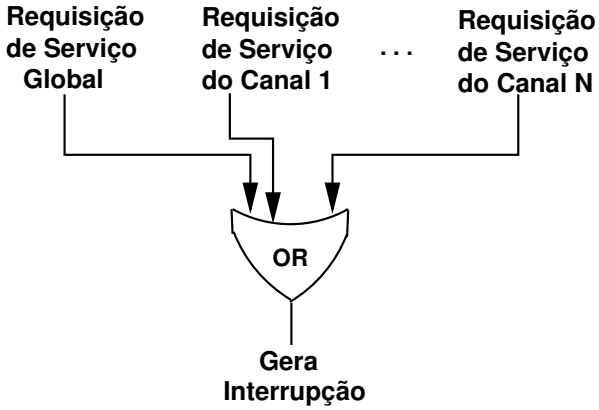

Figura 2.5: Geração de interrupção.

O *bit* de requisição de serviço é setado pela combinação dos *bits* de *status* padrão, dos *bits* de *status* auxiliar e das máscaras de interrupção. O sinal de interrupção é usado em conjunto com os registradores de *status* e de máscara de interrupção para indicar condições excepcionais no STIM.

O NCAP lerá tipicamente todos os outros registradores de *status* de canais para determinar qual canal ou quais canais estão requisitando serviços e para que fim. O NCAP não é obrigado responder a uma interrupção imediatamente. Além disso, o NCAP trata eventos de troca a quente (*hot swap*) antes de tratar os serviços de interrupção do STIM.

2.4.9 Principais características técnicas

Além do uso em redes de controle, os STIMs podem ser usados com microprocessadores em uma variedade de aplicações, tais como instrumentos portáteis e cartões de aquisição de dados (WOODS et al., 1996).

Representação de unidades físicas

O padrão IEEE 1451.2 adota um método geral para descrever unidades físicas medidas por sensores ou sob a ação de um atuador. Unidades físicas são codificadas em seqüências binárias de dez *bytes* e são representadas como um produto de sete unidades do Sistema Internacional de Unidades (SI) e duas unidades suplementares do SI, cada uma elevada a uma potência racional, conforme Tabela 2.3. Esta estrutura codifica apenas os expoentes. O produto está implícito.

As formas U/U expressam unidades adimensionais tais como deformação (metro por metro) e concentração (moles por mol). As unidades do numerador e do denominador são idênticas, com cada uma sendo especificada do campo dois ao campo dez da estrutura de tipos de dados para unidades físicas (IEEE, 1997; WOODS et al., 1996).

Tabela 2.3: Tipos de estruturas de dados de unidades físicas

N° do campo	**Descrição**	**N° de bytes**
1	ENUMERAÇÃO: 0. A unidade é descrita pelo produto de unidades base, adicionais radianos e *steradians* elevados às potências recordado no campo 2-10. 1. A unidade é da forma U/U, onde U é descrito pelo produto de unidades base, adicionais radianos e *steradians* elevados às potências recordado no campo 2-10. 2. A unidade é da forma $\log_{10} U$, onde U é descrito pelo produto de unidades base, adicionais radianos e *steradians* elevados às potências recordado no campo 2-10. 3. A unidade é da forma $\log_{10}(U/U)$, onde U é descrito pelo produto de unidades base, adicionais radianos e *steradians* elevados às potências recordado no campo 2-10. 4. Dado digital é uma quantidade associada a um vetor de *bit*. Este vetor de *bits* não tem unidade. O campo 2-10 deve ser setado para 128. O tipo de dado digital aplica-se ao dado que representa uma contagem, tais como a saída de um conversor AD, ou que representa um estado, tal como a posição atual de um banco de chaves. 5. A quantidade física associada é representada por valores sobre uma escala arbitrária. Os campos 2-10 são reservados e deve ser setado para 128. 6-255. Reservado	1
2	(2∗ <expoente em radianos>) + 128	1
3	(2∗ <expoente em steradians>) + 128	1
4	(2∗ <expoente em metros>) + 128	1
5	(2∗ <expoente em quilogramas>) + 128	1
6	(2∗ <expoente em segundos>) + 128	1
7	(2∗ <expoente em amperes>) + 128	1
8	(2∗ <expoente em kelvins>) + 128	1
9	(2∗ <expoente em moles>) + 128	1
10	(2∗ <expoente em candelas>) + 128	1

Modelo geral de calibração

O padrão IEEE 1451.2 fornece um modelo geral para especificar a opção de calibração do transdutor (BARFORD; LI, 1999; MOZEK et al., 2002). Ele é muito flexível, mas mesmo assim pode entrar em colapso (se tornar instável, impraticável) para uma simples relação linear de tamanho aceitável. A estrutura do modelo geral suporta um polinômio multivariável, suave por partes, com largura de segmento variável e *offset* de segmento variável (IEEE, 1997; WOODS et al., 1996).

Suporte para transdutores multivariável

O padrão 1451.2 inclui suporte para transdutores multivariável em um único STIM. Um STIM pode ter até 255 entradas ou saídas. Isso permite a criação de sensores e atuadores multivariável ou uma combinação de ambos (IEEE, 1997; WOODS et al., 1996).

2.5 TEDS

O TEDS (*Transducer Electronic Data Sheet*) é uma das principais inovações técnicas introduzidas pelo padrão IEEE 1451.2 (JOHNSON, 1997; WOODS et al., 1996). Ele carrega a informação sobre o transdutor e seu desempenho. Suporta uma grande variedade de transdutores com uma única estrutura de propósito geral, isto é, cada estrutura com um único campo requisitado, por exemplo um TEDS para estrutura do sensor de temperatura e outro para um servo atuador. Esta técnica facilita a implementação do restante do sistema tornando-o escalonável. Se campos específicos não são requisitados para um dado transdutor este campo tem largura nula, o que economiza memória (WOODS et al., 1996).

Os TEDSs contêm campos que descrevem completamente o tipo, a operação e os atributos do transdutor, tais como: data de fabricação, calibração específica (LEE; SCHNEEMAN, 1996) etc. Se o transdutor é movido para uma nova localização, o TEDS é movido com ele. Desta maneira a informação necessária para usar o transdutor no sistema está sempre presente.

Oito diferentes formatos de TEDS são definidos no padrão (IEEE, 1997; JOHNSON, 1997; LEE; SCHNEEMAN, 1996):

- **Meta-TEDS** - contêm o campo de dados que é comum a todos os transdutores conectados ao STIM. O campo contém informações tais como: descrição global da estrutura de dados do TEDS, parâmetros de tempo do STIM para o pior caso, etc;
- **TEDS de canal** - contêm informações de um transdutor específico no STIM. Cada transdutor tem seu TEDS de canal associado contendo informações tais como: unidades físicas, incertezas, limites superior e inferior de escala, etc;

- **TEDS de calibração** - contêm informações de parâmetros de calibração e intervalos de calibração;
- **TEDS de aplicação específica** - são para uso de aplicações específicas pelo usuário final;
- **TEDS de extensão** - são reservados para implementações futuras e extensão industrial do padrão;
- **TEDS de meta-identificação** - fornecem informações de identificação de dados para o STIM tais como: nome do fabricante, número do modelo, número serial, versão e data dos códigos dos transdutores, etc;
- **TEDS para identificação de canal** - possuem informações muito parecidas com a do TEDS de meta identificação, exceto que este é para um canal individual;
- **TEDS para identificação de calibração** - fornecem uma descrição de qualquer informação relevante à calibração de canal. Esta estrutura é repetida para cada canal e língua suportada.

A informação contida nos TEDS fornece capacidade de auto-identificação para transdutores que estão invalidados, fornece diagnósticos e determina características de tempo médio de operação antes de falhar.

Muito do conteúdo dos TEDS não precisam ser analisados e armazenados no NCAP. Isto dependerá da medição ou problema de controle que estiver sendo resolvido e da diferença do NCAP desejado. Se o aplicativo precisar do conteúdo do TEDS para usar em outra parte do sistema, então o TEDS pode ser lido pelo STIM que envia o conteúdo diretamente pela rede. Se o NCAP estiver fazendo medições no sensor e enviando leituras do mesmo, em unidades de engenharia, fatores de calibração devem ser analisados e armazenados no NCAP para ser usado para correção (WOODS et al., 1996).

2.6 Interface digital TII

A interface digital TII (*Transducer Independent Interface*) especifica a interface entre o STIM e o NCAP. Os protocolos, temporizações e especificações elétricas são definidas para assegurar transporte de dados robusto entre as diversas combinações de STIMs e NCAPs. Nessa seção são feitas diversas definições de cunho físico a respeito das linhas TII, suas funções, nomes e convenções lógicas de sinais.

2.6.1 Linhas

As linhas que compõem a interface que conecta o STIM ao NCAP são apresentadas em quatro grupos. Na Tabela 2.4 são listados os grupos, o nome completo das linhas e a abreviação padrão para cada linha.

Tabela 2.4: Linhas físicas TII

Grupo	**Linhas**	**Abreviação**
Dados	DATA_OUT	DOUT
	DATA_IN	DIN
	DATA_CLOCK	DCLK
	N_IO_ENABLE	NIOE
Gatilhamento	N_TRIGGER	NTRIG
Suporte	POWER	POWER
	COMMON	COMMON
	N_ACKNOWLEDGE	NACK
	N_STIM_DETECT	NSDET
Interrupção	N_IO_INTERRUPT	NINT

NOTA - NACK suporta transporte de dados e gatilhamento.

Convenções lógicas

Sinais digitais representam dados binários (0 ou 1) ou controle (*ASSERTED* - estado ativo ou *NEGATED* - estado inativo). As designações "verdadeiro" e "falso" são evitadas para impedir confusões causadas quando eles são associados com níveis específicos de sinais. Fisicamente representados por uma das duas tensões definidas nas especificações elétricas, os sinais digitais com suas terminologias e associações são definidos nas Tabelas 2.5 e 2.6.

Tabela 2.5: Terminologia para níveis lógicos

Terminologia	**Nível de tensão**	**Significado associado**
Lógica Positiva, Ativa *High*, *High Asserted*	*LOW*	Binário zero ou sinal *NEGATED*
	HIGH	Binário um ou sinal *ASSERTED*
Lógica Negativa, Ativa *Low*, *Low Asserted*	*HIGH*	Binário zero ou sinal *NEGATED*
	LOW	Binário um ou sinal *ASSERTED*

Eventos são usualmente associados com transições de um nível lógico para outro. Uma transição *LOW-to-HIGH* é chamada de transição de borda positiva e a transição *HIGH-to-LOW* deve ser chamada de transição de borda negativa.

Sinais que são ativos *LOW* ou são associados com bordas negativas têm nome pré-fixados com o mnemônico auxiliar N.

Tabela 2.6: Terminologia para borda do sinal

Terminologia	**Significado associado**
Borda Positiva, Borda de Subida	Transição de *LOW-to-HIGH*
Borda Negativa, Borda de Descida	Transição de *HIGH-to-LOW*

Definição das linhas

As linhas físicas da interface para o transdutor são descritas brevemente na Tabela 2.7.

Tabela 2.7: Sinal das linhas da interface

Linha	**Lógica**	**Guiado por**	**Função**
DIN	Positiva	NCAP	Transporte de dados e endereços do NCAP para o STIM
DOUT	Positiva	STIM	Transporte de dados do STIM para o NCAP
DCLK	Borda positiva	NCAP	A troca de dados é feita na borda positiva sobre ambos DIN e DOUT
NIOE	Ativo baixo	NCAP	Sinais que transportam dados são ativos e delimitam os moldes do transporte de dados
NTRIG	Borda negativa	NCAP	Desempenha função de gatilhamento
NACK	Borda negativa	STIM	Apresenta duas funções: reconhecimento de gatilhamento e reconhecimento de transporte de dados
NINT	Borda negativa	STIM	Usado pelo STIM para requisitar serviços do NCAP
NSDET	Ativo baixo	STIM	Usado pelo NCAP para detectar a presença de um STIM
POWER	N/A	NCAP	Fonte de potência nominal de 5 V
COMMON	N/A	NCAP	Sinal comum ou terra

Protocolos

Os protocolos descrevem a implementação das funções de gatilhamento e transporte de dados usando implementação física TII. Ambos, o NCAP e o STIM, participam de cada protocolo e suas regras individuais são identificadas. Todos os protocolos são hierarquicamente definidos. Os protocolos superiores são para leitura e escrita de quadros (*frame*),

além de gatilhamento. A seqüência de leitura e escrita entre o NCAP e o STIM é ilustrada na Figura 2.6.

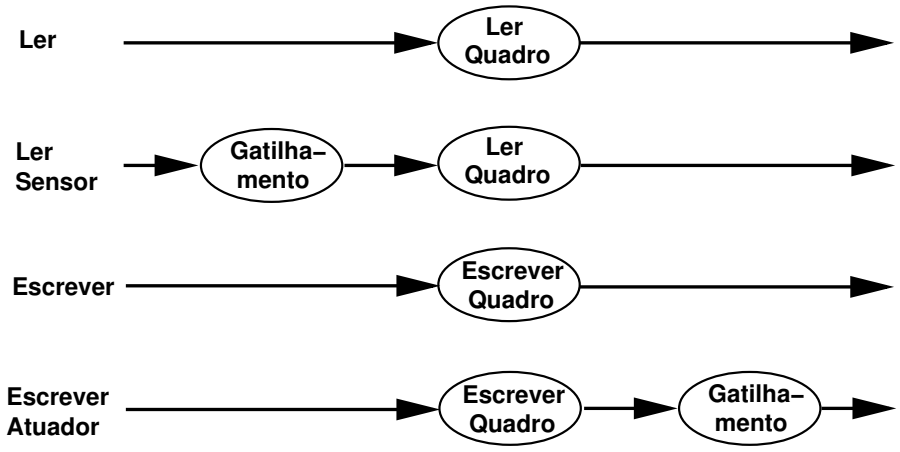

Figura 2.6: Protocolo geral de transferência de dados.

Quadros são uma seqüência de transferência de *bytes*. Transferência de *byte* é uma seqüência de transferência de *bits*.

O estado inicial para todo gatilhamento, leitura e escrita de quadros ocorre com as linhas NTRIG, NACK e NIOE negadas.

Gatilhamento

O gatilhamento é normalmente usado antes da leitura de um sensor e depois da escrita de um atuador.

a) O NCAP espera pela duração do tempo de execução de escrita do canal (relembrado no TEDS de canal);

b) O NCAP declara NTRIG;

c) O STIM declara NACK;

d) O NCAP nega NTRIG;

e) O STIM nega NACK;

f) O NCAP espera pela duração do tempo de execução de leitura do canal (relembrado no TEDS de canal).

Transferência de Bit

Dados são transferidos na forma de *bit* serial do NCAP para o STIM via DIN e do STIM para o NCAP via DOUT. A transferência é controlada pela linha DCLK da seguinte maneira:

a) DCLK está inativo em nível alto (*DCLK idles high*);

b) Durante a primeira borda de descida de DCLK, o primeiro *bit* a ser transferido é declarado pelo emissor (o NCAP sobre DIN e o STIM sobre DOUT);

c) Na borda de subida subseqüente de DCLK, o *bit* é armazenado pelo receptor (o STIM sobre DIN e o NCAP sobre DOUT). Esta borda de subida tem que estar de acordo com a temporização;

d) Os *bits* subseqüentes são transferidos pela repetição dos passos b) e c).

Embora em uma transmissão *full-duplex* seja possível usar DIN e DOUT simultaneamente, esta especificação não faz uso de tal característica. Isto é, quando dados são transferidos do NCAP para o STIM a linha DOUT é ignorada pelo NCAP. Por outro lado, quando dados são transferidos do STIM para o NCAP o STIM ignora a linha DIN.

NOTA - DCLK não precisa ter uma freqüência constante ou duty cycle.

Protocolo de transferência para escrita de *byte* do NCAP para STIM

Todos os dados devem ser transferidos do NCAP para o STIM em grupos de 8 *bits* usando o protocolo de transferência de *bit*. O NCAP só deve proceder com uma transferência de um *byte* para escrita depois que ele observar a transição da linha NACK. O STIM deve causar uma transição sobre a linha NACK quando ele for corretamente controlado pelo *byte* prévio e estiver pronto para o NCAP prosseguir.

Protocolo de transferência para leitura de *byte* do STIM para NCAP

Todos os dados devem ser transferidos do STIM para o NCAP em grupos de 8 *bits*, usando o protocolo de transferência de *bit*. O NCAP só deve proceder com uma transferência de um *byte* para leitura depois que ele observar a transição da linha NACK. O STIM deve causar uma transição sobre a linha NACK quando ele for corretamente controlado pelo *byte* prévio e estiver pronto para o NCAP prosseguir.

Protocolo de transferência para leitura de quadro

O protocolo para leitura do quadro é ilustrado na Figura 2.7 e 2.8 e explanado como se segue:

A) O NCAP declara NIOE;

B) O NCAP espera o STIM declarar NACK;

C) O NCAP escreve o endereço funcional usando o protocolo de transferência para escrita de *byte*;

D) O NCAP escreve o endereço do canal usando o protocolo de transferência para escrita de *byte*;

E) O NCAP lê zero ou mais *bytes* de dados usando o protocolo de transferência para leitura de *byte*, do *byte* mais significativo até o *byte* menos significativo;

F) O NCAP nega NIOE;

G) O STIM nega NACK. (Se um número ímpar de *bytes* foi transferido, NACK já estará negado devido o protocolo de transferência para leitura de *byte*).

Protocolo de transferência para escrita de quadro

O protocolo para escrita de quadro é ilustrado na Figura 2.7 e 2.8, e explicado como se segue:

A) O NCAP declara NIOE;

B) O NCAP espera o STIM declarar NACK;

C) O NCAP escreve o endereço funcional usando o protocolo de transferência para escrita de *byte*;

D) O NCAP escreve o endereço do canal usando o protocolo de transferência de *byte* para escrita;

E) O NCAP escreve zero ou mais *bytes* de dados usando o protocolo de transferência para escrita de *byte*, do *byte* mais significativo até o *byte* menos significativo;

F) O NCAP nega NIOE;

G) O STIM nega NACK. (Se um número ímpar de *bytes* foi transferido, NACK já será negado devido o protocolo de transferência para leitura de *byte*).

Protocolo de transporte de dados e notas explicativas

Nas Figuras 2.7 e 2.8 são mostrados quadros de transporte de dados para um número par e ímpar de *bytes* respectivamente. As letras maiúsculas referem-se aos protocolos de leitura e escrita de quadros. Nos asteriscos são destacadas as transições NACK.

Ao final do quadro NACK estará declarada ou negada, dependendo se o número de *bytes* transferidos for ímpar ou par respectivamente. Se NACK estiver declarada ao fim de

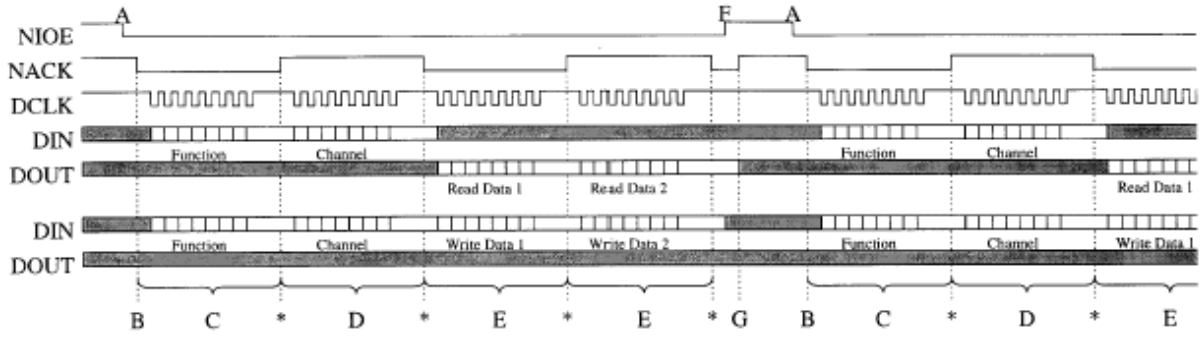

Figura 2.7: Quadro de transporte de dados com número par de *bytes* transferidos

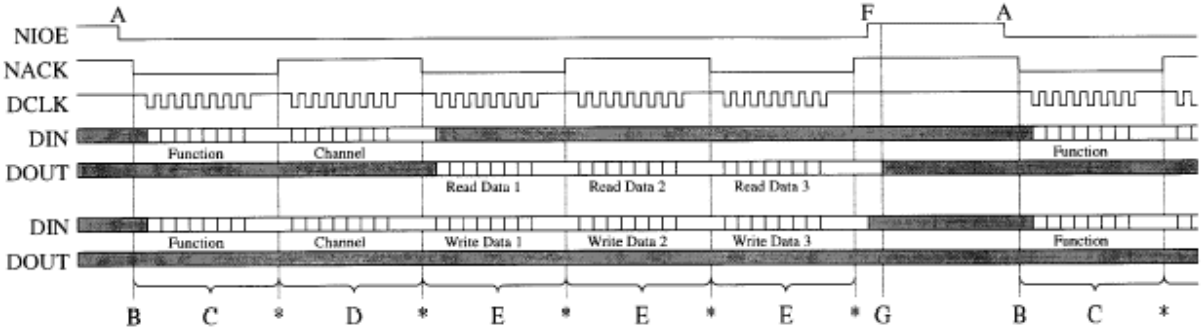

Figura 2.8: Quadro de transporte de dados com número ímpar de *bytes* transferidos

um quadro de transporte de dados, então tem-se uma ação de *handshake* para a negação de NIOE. Se NACK estiver negada durante a negação de NIOE, então o NCAP pode se assegurar que o STIM reconheceu o fim do quadro de transporte de dados, esperando apenas a duração do tempo de latência para a detecção do final do quadro antes de re-declarar NIOE.

Exceções

NTRIG e NIOE nunca devem ser declaradas ao mesmo tempo. Se isso acontecer o STIM pode se comportar de maneira imprevisível. O transporte de dados e o mecanismo de gatilhamento podem ser resetados pelo NCAP, negando NTRIG e NIOE, e o STIM deve responder negando NACK (se ele não estiver pronto para terminar).

O transporte do quadro de dados requer que NIOE seja declarado e o NCAP pode negar NIOE antes de uma completa estrutura de dados ser transmitida, abortando o quadro. O NCAP deve negar NIOE apenas entre *bytes*. Se NIOE é negada em qualquer instante durante a função de transporte de dados então o STIM deve negar NACK (se ele não estiver pronto para terminar) e resetar o mecanismo de transporte de dados. Se a duração da negação do NIOE apresentar ou exceder a detecção de latência no fim do quadro o STIM deverá está pronto para detectar outra declaração de NIOE, a qual o

STIM deve interpretar como o início de uma nova função de transporte de dados.

Se o STIM permanecer sem transporte de dados por um tempo longo, mesmo que satisfaça os tempos apropriados de *hold-off*, especificado no TEDS, o NCAP pode assumir que o transporte de dados está em condições errôneas e abortá-lo, negando NIOE. Se o NCAP declarar NTRIG e depois negá-lo, antes do STIM declarar NACK, o gatilhamento deve ser abortado. Se o tempo entre a declaração de NTRIG e a declaração de NACK exceder o tempo de atualização do canal, o NCAP pode assumir que o transporte de dados está sob condições errôneas e abortá-lo, negando NTRIG.

Temporização

A taxa de transporte de dados, à freqüência nominal para DCLK, pode ser variada pelo NCAP. Todos os NCAPs e STIMs devem suportar uma taxa de dados comum de 6000 $bits/s$, que deve ser a taxa de comunicação até que o NCAP leia o Meta-TEDS. Depois que o NCAP ler o *Meta-TEDS* ele pode chavear para uma taxa de dados menor ou igual à taxa de dados máxima especificada pelo *Meta-TEDS*. O NCAP e o STIM devem suportar uma taxa de dados de no mínimo 6000 $bits/s$. O STIM deve reabilitar a comunicação em qualquer taxa de dados, inferior ao seu máximo especificado. Interrupções de DCLK por longos períodos não devem interromper a comunicação do STIM.

A temporização do transporte de dados é complicada pelo fato que o NCAP permite mudar a taxa de dados. Para assegurar o transporte de dados na taxa selecionada, muitos parâmetros de temporização relacionados com o transporte de dados, tais como tempos de execução e permanência, são baseados em um dos seguintes parâmetros:

a) f_{max} - a máxima taxa de dados suportado pelo STIM é especificada no *Meta-TEDS*;

b) f_{clk} - a taxa de dados selecionada pelo NCAP. DCLK atinge esta taxa;

c) f_{clk_max} - a máxima taxa de dados que o NCAP é capaz de suportar;

d) f_{clk_min} - a mínima taxa de dados que o NCAP é capaz de suportar.

Na Tabela 2.8 são listados os requisitos de temporização do NCAP. Veja a Figura 2.9 para definições dos parâmetros de temporização do NCAP.

Os três primeiros requisitos na Tabela 2.8 aplicam-se ao sistema operacional, no qual o NCAP está mudando a taxa de dados e assim, direta ou indiretamente, também mudando os tempos alto e baixo de DCLK. Os tempos mínimos, alto e baixo de DCLK são suficientes para assegurar que todos os tempos de execução e permanência são apresentados quando o STIM está diretamente conectado ao NCAP, desde que todos os outros requisitos na Tabela 2.8 e 2.9 sejam apresentados individualmente pelo STIM e NCAP.

Dentro das restrições de temporização DCLK não precisa ter uma freqüência constante ou *duty cycle*.

Tabela 2.8: Requisitos de temporização do NCAP

Características	**Símbolo**	**Unidade**	**Requisito**
taxa DCLK	f_{clk}	bits por segundo	$f_{clk} \leq f_{max}$
tempo alto DCLK	t_{ch}	segundos	$t_{ch} \geq \frac{0,2}{f_{clk}}$
tempo baixo DCLK	t_{cl}	segundos	$t_{cl} \geq \frac{0,45}{f_{clk}}$
tempo de execução DCLK para validar DOUT	t_{sun}	segundos	$t_{sun} \leq \frac{0,2}{f_{clk\_max}}$
tempo de permanência inválido de DCLK para DOUT	t_{hn}	segundos	$t_{hn} \leq \frac{0,2}{f_{clk\_max}}$
atraso válido de DIN para DCLK	t_{dn}	segundos	$t_{dn} \leq \frac{0,2}{f_{clk\_max}}$
tempo de subida ou descida: DCLK	t_{edgex}	segundos	$t_{edgex} \leq \frac{0,05}{f_{clk\_max}}$
tempo de subida ou descida: DCLK, DIN, NIOE, NTRIG	t_{edgen}	microssegundos	$t_{edgen} \leq 2$

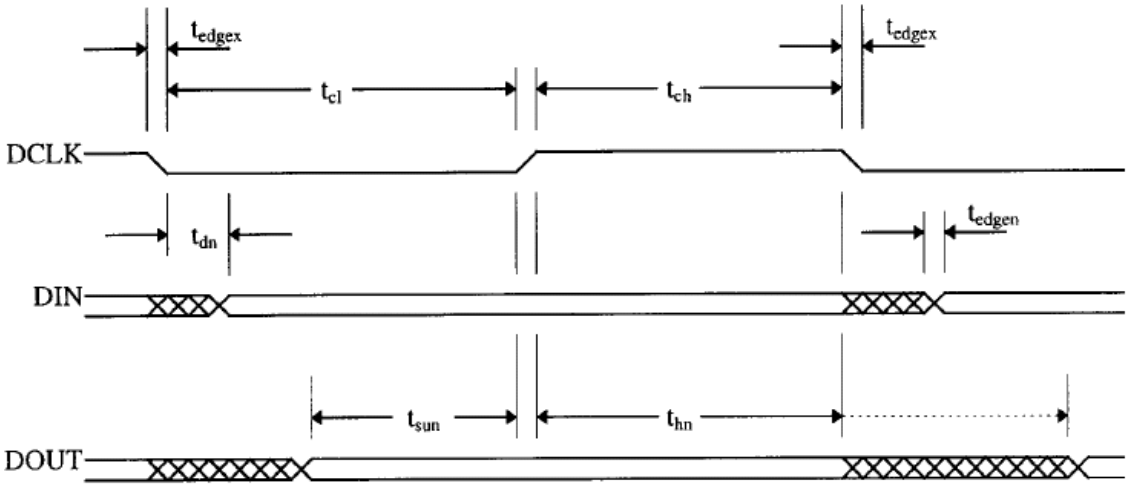

Figura 2.9: Diagrama de temporização das linhas de dados do NCAP

Os fabricantes de NCAPs podem determinar t_{sun}, t_{hn}, t_{dn}, t_{edgen} e t_{edgex}, medindo o NCAP no ponto de conexão TII. Tempos de execução e permanência incluem atrasos de filtragem ou circuitos de proteção sobre as entradas e saídas do NCAP. DCLK, DIN, NIOE e NTRIG devem ser carregadas por uma capacitância de até 200 pF. DOUT deve ser alimentado por um gerador capaz de produzir tempos de subida e descida menores que $0,01/f_{clk\_max}$ segundos. Os fabricantes dos NCAPs devem escolher f_{clk_max} tal que, todos os requisitos da Tabela 2.8 sejam apresentados.

Na Tabela 2.9 são listados os requisitos de temporização para o STIM. Veja Figura 2.10 para as definições dos parâmetros de temporização do STIM.

Tabela 2.9: Requisitos de temporização do STIM

Características	**Símbolo**	**Unidade**	**Requisito**
máxima taxa de dados	f_{max}	*bits* por segundo	$f_{max} \geq 6000$
DIN válido para tempo de execução DCLK	t_{su}	segundos	$t_{su} \leq \frac{0,2}{f_{max}}$
tempo de permanência inválido de DCLK para DIN	t_h	segundos	$t_h \leq \frac{0,2}{f_{max}}$
atraso válido de DCLK para DOUT	t_d	segundos	$t_d \leq \frac{0,2}{f_{max}}$
tempo de subida ou descida: DOUT, NACK, NINT	t_{edges}	microssegundos	$t_{edges} \leq 2$

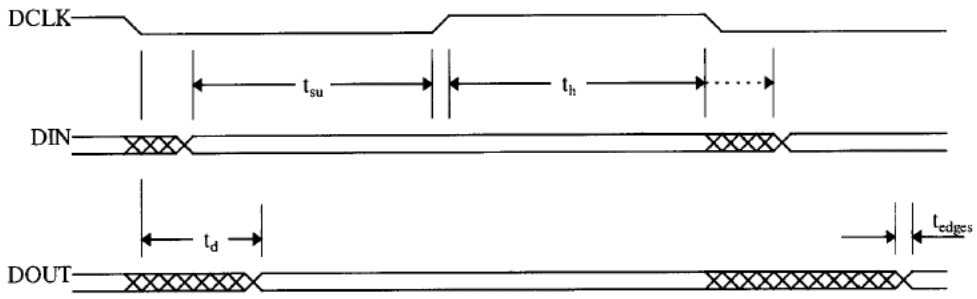

Figura 2.10: Diagrama de temporização das linhas de dados do STIM

Os fabricantes de STIMs devem determinar t_{su}, t_h, t_d e t_{edges}, medindo o ponto de conexão do STIM ao TII. Tempos de execução e permanência incluem atrasos de filtragem ou circuitos de proteção sobre as entradas e saídas do STIM. DOUT, NACK e NINT devem ser carregadas por uma capacitância de até 200 pF. DCLK e DIN devem ser alimentados por um gerador capaz de produzir tempos de subida e descida menores que $0,01/f_{clk\_max}$ segundos. Os fabricantes dos STIMs podem escolher f_{clk_max} tal que, todos os requisitos da Tabela 2.9 sejam satisfeitos.

NOTA - os requisitos das Tabelas 2.8 e 2.9 partem de uma estimativa de $0,05/f_{clk}$ segundos para incertezas e atrasos de cabos. Isto praticamente limita um sistema sem cabeamento e com incertezas de temporização de 5 ns para uma taxa de dados máxima de 10 Mb/s. Um sistema com cabo, com atrasos de ida e volta de 50 ns, está limitado praticamente a uma taxa de dados máxima de 1 Mb/s.

Os diagramas de temporização nesta subcláusula relatam os parâmetros de temporização encontrados no Meta-TEDS e TEDS de canais.

O *handshake*[1] de temporização do STIM ocorre devido a negação de NACK durante a exucução do protocolo de gatilhamento e durante o transporte de dados, como ilustrado nas Figuras 2.11 e 2.12. Observe na Figura 2.11 que a linha NACK pode ser negada antes que NIOE seja negada. Entretanto, NACK deve ser negada sem atraso, mesmo que NIOE seja negada t_{hs} depois.

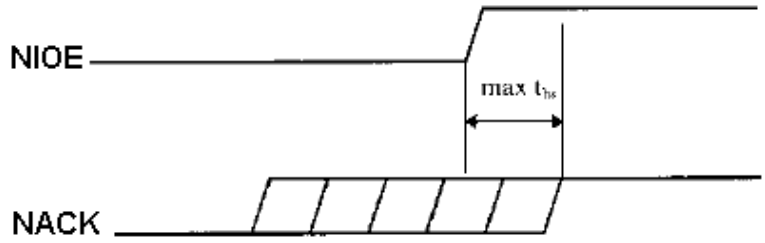

Figura 2.11: Fim do quadro de temporização

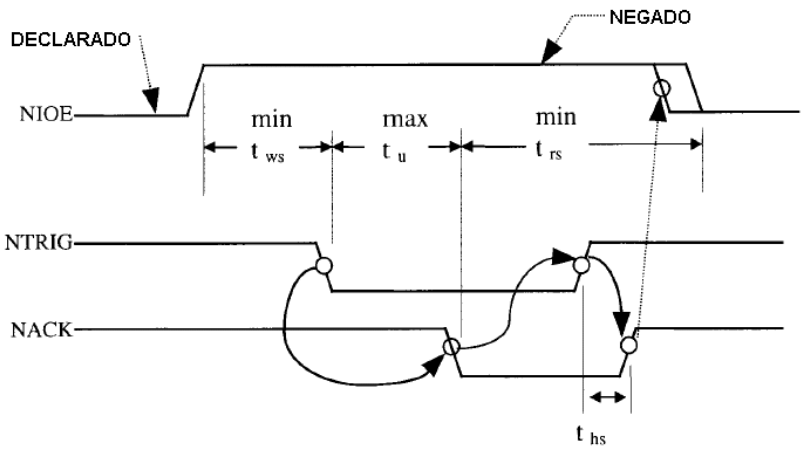

Figura 2.12: Temporização de gatilhamento

Na Figura 2.12 também são ilustradas e definidas as relações de temporização entre o transporte de dados envolvendo o canal gatilhado e o gatilhamento daquele canal. Para não confundir nenhum outro transporte de dados é mostrado.

a) t_{ws} mínimo deve ser o tempo de execução de escrita do canal especificado no TEDS de canal, se dados têm sido escritos para o canal do atuador ou zero em caso contrário. Se o gatilhamento está sobre o canal zero t_{ws} deve ser trocado por t_{gws} , o tempo de execução de escrita global é especificado no Meta-TEDS;

b) t_{rs} mínimo deve ser o tempo de execução de leitura do canal especificado no TEDS de canal quando um canal de sensor for gatilhado. Se nenhuma leitura do canal

[1] Tempo decorrido entre a mudança do estado da linha NIOE e a mudança de estado da linha NACK, para reconhecimento por parte do STIM, da mudança do estado de NIOE, pelo NCAP

está para ser executada seguindo o gatilhamento então NIOE pode ser declarado tão logo quanto NACK seja negado. Se o gatilhamento está sobre o canal zero então t_{rs} deve ser trocado por t_{grs} , o tempo de execução de leitura global é especificado no Meta-TEDS;

c) t_u máximo deve ser o tempo de atualização do canal especificado no TEDS de canal, quando um canal for gatilhado. Se o gatilhamento está sobre o canal zero t_u deve ser trocado por t_{wu} , o pior caso do tempo de atualização de canal, especificado no Meta-TEDS.

Na Figura 2.13 são ilustradas as definições das restrições de temporização, quando um canal previamente gatilhado é novamente gatilhado. Esta temporização deve ser aplicada a qualquer canal individual, indiferentemente de qualquer atividade de transporte de dados ou gatilhamento de qualquer outro canal.

a) t_{sp} mínimo deve ser o período de amostragem do canal especificado no TEDS de canal. Se o gatilhamento está sobre o canal zero t_{sp} deve ser trocado por t_{wsp} , o pior caso do período de amostragem especificado no Meta-TEDS;

b) Não obstante a quaisquer outras considerações, t_{sp} ou t_{wsp} está restringido pelos tempos de execução de *handshake*, como segue:

$$t_{sp} \geq t_u + h_{hs} \quad ou \quad t_{wsp} \geq t_{wu} + h_{hs}$$

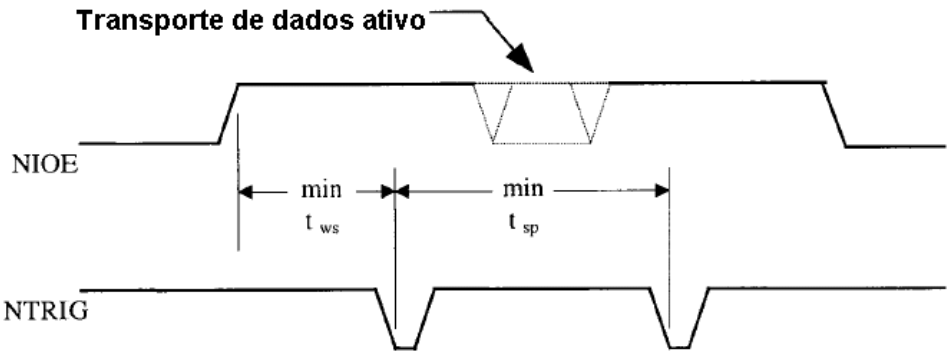

Figura 2.13: Temporização de múltiplos NTRIG

Na Figura 2.14 é ilustrada a definição do tempo de aplicação para detecção de latência do fim do quadro (t_{lat}) no Meta-TEDS. Negando NIOE um mínimo de t_{lat} segundos entre quadros de transporte de dados, o NCAP pode assegurar que o STIM possa detectar o fim de um quadro e o início de outro.

O tempo de *hold-off* do *byte*, t_{hold}, mostrado na Figura 2.15, é o tempo entre a última borda de subida de DCLK de um *byte* transferido e a transição NACK realizada pelo

Figura 2.14: Latência para transporte de dados

STIM durante o transporte de dados. O máximo t_{hold} deve ser o tempo de *hold-off* do TEDS se o transporte de dados é endereçado ao TEDS de funções ou tempo operacional de *hold-off* se o transporte de dados é endereçado a funções operacionais. Se o transporte de dados é endereçado ao canal de dados do transdutor, a soma de todos os *hold-off* para o transporte de dados do quadro inteiro não deve exceder o tempo de *hold-off* dos canais juntos do TEDS de canal. Se os dados do transdutor são endereçados ao canal zero, a soma de todos os *hold-off* não deve exceder a soma de todos os tempos de *hold-off* dos canais juntos. Se quaisquer destas restrições de tempo de *hold-off* forem violadas, o NCAP pode assumir que a função de endereço não está trabalhando adequadamente e pode abortar o quadro de transporte de dados.

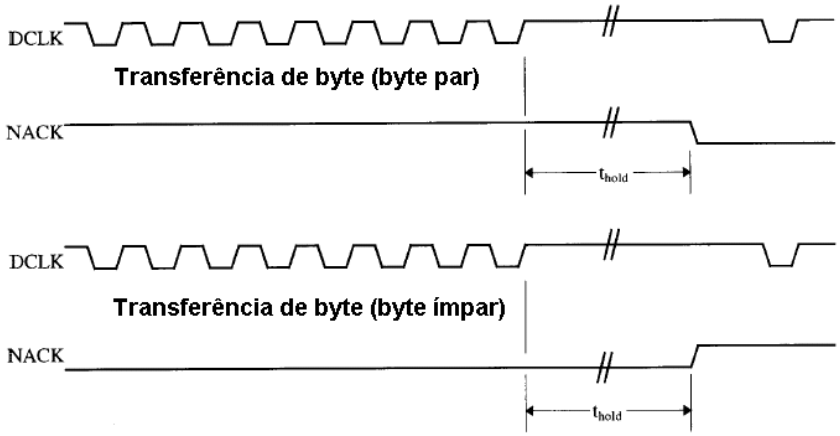

Figura 2.15: Temporização do *byte* de *hold-off*

2.6.2 Especificações elétricas

Alimentação

a) A tensão sobre a linha POWER do NCAP deve ser de 5 $V\ dc$ $\pm 0,20\ V\ dc$ com respeito ao comum (COMMON);

b) O fabricante de cada STIM deve especificar a potência que ele consome. O STIM não deve contar com mais que 75 mA na tensão especificada do NCAP (5 $V\ dc$ $\pm 0,20$ $V\ dc$);

c) O STIM pode usar uma fonte de tensão separada quando necessário. Entretanto, a alimentação para os circuitos de controle da interface STIM, tipicamente consistindo de um microprocessador, FPGA (*Field Programmable Gate Array*), ou ASIC (*Application-Specific Integrated Circuit*) deve ser fornecida apenas através da interface de comunicação primária (o TII);

d) Em todos os casos os sinais das linhas de potência e comum do TII devem ser isolados da terra (estrutura ou neutro);

e) Um STIM não deve ser capaz de ativar fontes de corrente sobre a linha POWER;

f) É altamente recomendado, mas não requisitado, que o NCAP possa controlar ativamente a aplicação de potência para o STIM. Neste caso, o NCAP deve fornecer potência (tipicamente através de uma chave FET) apenas quando o STIM estiver presente.

Na Figura 2.16 é ilustrado o controle ativo da aplicação de potência ao STIM pelo NCAP. Neste caso, a potência é fornecida pelo NCAP apenas se o STIM estiver presente (eliminando a possibilidade de curtos circuitos). Note que tipicamente pode existir algum atraso curto antes de um evento de inserção ou de extração ser reconhecido pelo NCAP.

Os seguintes eventos são mostrados na Figura 2.16:

A) Ocorre um evento de inserção;

B) O NCAP ativa a potência para o STIM;

C) Ocorre um evento de extração;

D) O NCAP retira a potência do STIM;

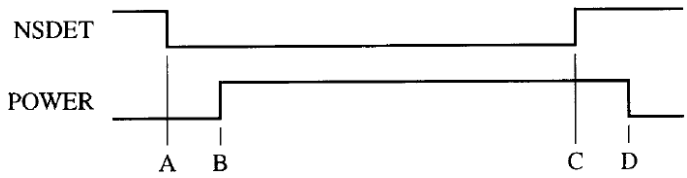

Figura 2.16: Controle ativo de potência do STIM

O controle não-ativo de potência do NCAP para o STIM é ilustrado na Figura 2.17. Neste caso, a potência é entregue imediatamente ao STIM simultaneamente à declaração de NSDET. Os seguintes eventos são evidenciados pela Figura 2.17:

E) Ocorre um evento de inserção e é disponibilizada potência para o STIM;

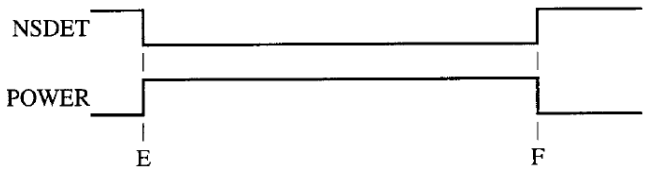

Figura 2.17: Controle não-ativo de potência do STIM

F) Ocorre a extração de evento e é retirada a potência do STIM.

Na Figura 2.18 é ilustrada a configuração na qual o NCAP fornece potência aos circuitos de controle da interface STIM e a quaisquer circuitos de medição e/ou atuação. O STIM não precisa ser afetado quando o NCAP retira a potência.

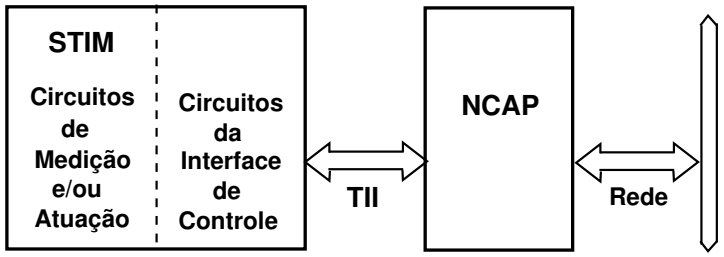

Figura 2.18: O NCAP alimenta os circuitos de controle do STIM e o circuito STIM mede e/ou atua

Na Figura 2.19 é ilustrada a configuração na qual o NCAP fornece potência aos circuitos da interface de controle do STIM, entretanto a potência necessária para os circuitos de medição e/ou atuação excede os níveis de corrente e/ou tensão que pode ser fornecida pela interface de comunicação primária. Neste caso, a potência para os circuitos de medição e/ou atuação pode ser fornecida por uma fonte de potência secundária. O STIM não precisa ser afetado pela fonte de potência secundária opcional.

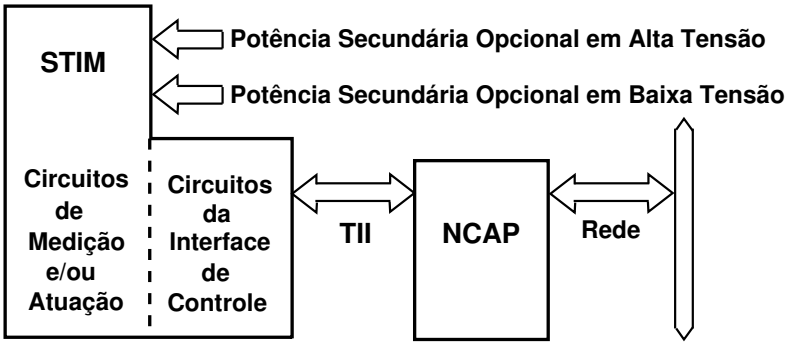

Figura 2.19: O NCAP alimenta os circuitos de controle do STIM. O conector secundário opcional alimenta os circuitos de medição e/ou atuação do STIM

Sinais lógicos

Nesta seção são especificados os níveis lógicos, energização e características de carregamento que devem ser aplicados aos sinais lógicos designados por DOUT, DIN, DCLK, NIOE, NTRIG, NACK e NINT. Estas especificações são baseadas na lógica CMOS de 5 $V\ dc$ nominal.

Níveis lógicos

Em qualquer sinal de entrada sobre o NCAP ou STIM, uma tensão maior que $0,8$ vezes a tensão da linha de potência (*POWER*) pode ser interpretado como lógica alta (*HIGH*) e uma tensão menor que $0,15$ vezes a tensão do nó de potência (*POWER*) deve ser interpretada como lógica baixa (*LOW*).

NOTA - A lógica CMOS comum admite limites percentuais de 70% e 30% da tensão fornecida pela fonte e a lógica TTL comum admite limites de tensão de 2 $V\ dc$ e $0,8\ V\ dc$ que cumprem prontamente com esses requisitos.

Energização e carregamento

a) Entrada e saída de cargas são definidas em termos de tensões específicas para assegurar compatibilidade. Saídas são assumidas dentro de uma faixa de tensão definida pelos nós *POWER* e *COMMOM*. O padrão IEEE 1451.2 não define as tensões permitidas fora dessa faixa de entrada;

b) Qualquer saída levada a nível lógico alto deve ser capaz de fornecer um mínimo de 200 μA em tensões de saída menor que $0,9$ vezes a tensão do nó *POWER*;

c) Qualquer entrada não deve consumir uma corrente superior a 20 μA, mesmo que as tensões de saída sejam maiores que $0,9$ vezes a tensão do nó *POWER*;

d) Qualquer saída levando a um nível lógico baixo deve ser capaz de consumir um mínimo de $1,5\ mA$, em tensões de saída maiores que $0,1$ vezes a tensão do nó POWER;

e) Qualquer entrada deve fornecer não mais que 600 μA em tensões menores que $0,1$ vezes a tensão do nó POWER;

f) Saídas do NCAP e STIM devem apresentar especificações de temporização que levem a capacitâncias de até 200 pF;

g) Entradas do NCAP e STIM devem apresentar um máximo de 100 pF de capacitância;

h) Qualquer cabo entre o NCAP e o STIM não deve exceder 100 pF de capacitância entre quaisquer dois fios ou entre qualquer fio e a proteção (se uma proteção é usada).

Estados quiescentes

Linhas de sinais de entrada do NCAP - NINT, NACK, DOUT, e NSDET - devem permanecer passivamente altas, devido a uma grande resistência (tipicamente maior que 10000 Ω) para POWER de modo que estes sinais assumam um nível lógico conhecido se nenhum STIM está conectado ao NCAP. A corrente imposta por essa resistência deve está de acordo com a corrente de entrada fornecida pelas especificações fornecidas na seção 2.6.2.

Capacidade de troca a quente (*Hot-swap*)

Detecção de eventos de inserção ocorre quando o STIM é conectado ao NCAP energizado e a detecção de eventos de extração ocorre quando o STIM é desconectado do NCAP energizado. Estes eventos são sinalizados usando o sinal NSDET. No STIM NSDET deve ser conectado ao comum (terra). No NCAP, NSDET deve ser suavemente levado a POWER e deve ser controlada pelo processador do NCAP. Os estados do sinal NSDET são mostrados na Tabela 2.10.

Tabela 2.10: Estados do sinal NSDET

Evento	**NSDET**	**Ação**
Inserção	Torna-se declarado	levado para baixo
Extração	Torna-se negado	levado para alto

Todas as linhas de sinal do IEEE 1451.2 que são comandadas pelo STIM, exceto NSDET, devem ser negadas durante a inicialização do STIM. Um evento de inserção é detectado pelo NCAP, a qualquer instante que NSDET é declarado e um evento de extração é detectado a qualquer instante que NSDET é negado. Durante a conexão mecânica do NCAP ao STIM, o sinal sobre NSDET pode oscilar e pode precisar algum tempo para estabilizar.

Depois que POWER é aplicado ao STIM, o STIM precisa de um período de tempo para iniciar a si mesmo. A inicialização do STIM, neste contexto, refere-se ao STIM chegar ao ponto onde ele possa se comunicar com o NCAP - aquecimento de transdutores é um tipo diferente de atraso. O NCAP pode declarar NIOE a qualquer instante depois que NSDET for estabelecido no estado declarado e NACK for estabilizado no estado negado. Assim o NCAP não pode continuar a transição até que NACK seja declarado, o NCAP deve esperar até que a inicialização do STIM seja completada (assim todas as linhas de sinal IEEE 1451.2 guiadas pelo STIM são negadas durante a inicialização do STIM) e o STIM é capaz de responder ao NIOE com NACK. O NCAP pode então proceder a comunicação com o STIM na taxa de dados mínima suportada pelo STIM. No fim da transição, o NCAP nega NIOE e espera até que o STIM, em resposta, negue NACK.

Na Figura 2.20 é ilustrado o mecanismo pelo qual o STIM comunica ao NCAP que a inicialização do STIM foi completada. Os seguintes eventos ocorrem durante a inicialização do STIM:

A) Ocorre um evento de inserção;

B) O NCAP inicia uma transação declarando NIOE;

C) A inicialização do STIM está completa;

D) O STIM declara NACK em resposta à declaração de NIOE;

E) O NCAP nega NIOE para sinalizar o fim da transação;

F) O STIM nega NACK em resposta à negação de NIOE.

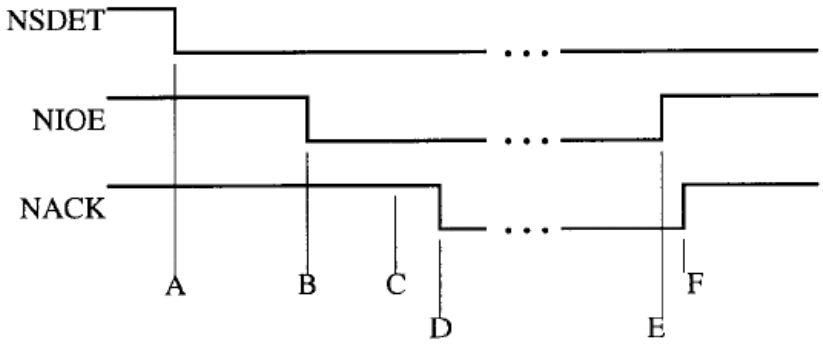

Figura 2.20: Início da comunicação temporizada do STIM

2.6.3 Especificações físicas

Aqui são descritas as especificações físicas de conectores para o TII, definido o código de cores dos fios usados na *interface* TII. Além de discutir a conexão de fontes de alimentação secundárias para o STIM.

Conectores

A numeração dos pinos do conector mostrada na Tabela 2.11 é recomendada. Também é mostrado um código de cores recomendado para STIMs não terminados.

Conector de comunicação primária

No padrão IEEE 1451-2 há ilustração dos conectores da conexão física TII. Entretanto é recomendado que o conector do STIM seja:

a) De no mínimo dez pinos;

b) Macho (pois o STIM não fornecerá potência);

c) Polarizado.

Tabela 2.11: Especificação dos pinos do conector TII para comunicação primária

Número do Pino	**Nome do Sinal**	**Cor do fio**	**Direção para o NCAP**	**Direção para o STIM**
1	DCLK	marrom	OUT	IN
2	DIN	vermelho	OUT	IN
3	DOUT	laranja	IN	OUT
4	NACK	amarelo	IN	OUT
5	COMMON (GROUND)	verde	POWER	POWER
6	NIOE	azul	OUT	IN
7	NINT	violeta	IN	OUT
8	NTRIG	cinza	OUT	IN
9	POWER ($+5\ V\ dc$)	branco	POWER	POWER
10	NSDET	preto	IN	OUT

Conector de potência secundário opcional

É recomendado que um conector secundário opcional para o STIM seja:

a) Macho, pois o STIM não fornecerá potência;

b) Polarizado.

2.7 NCAP

O IEEE 1451.1, Informação do Modelo do NCAP (IEEE, 1999), trata da definição de um modelo de objeto comum para componentes de uma rede de transdutores inteligentes e as especificações de *software* para interfacea-los (LEE, 1999; LEE; SCHNEEMAN, 1996; SCHNEEMAN, 1999).

A definição de NCAP, também chamado nó de rede (SCHNEEMAN, 1999), engloba um conjunto de classes de objetos, atributos, métodos (GOMAA, 2000; PASCHOA, 2001; LISBôA, 2003) e comportamentos que fornece uma descrição concisa do transdutor e da

rede a qual ele está conectado (LEE; SCHNEEMAN, 1996; SCHNEEMAN, 1999). Pela modelagem do dispositivo transdutor em termos orientado a objeto (POOLEY, 2000), uma visão abstrata das características do dispositivo podem ser reunidas em um modelo singular, suficientemente geral para abranger uma larga variedade de aplicações de serviços para redes de transdutores (SCHNEEMAN, 1999). O modelo de objeto trata dos dois problemas específicos da área. A padronização da integração entre, como aplicações interagem com sensores e atuadores físicos no sistema e como esta mesma aplicação interage com a rede, em que está conectada (LEE; SCHNEEMAN, 1996; SCHNEEMAN, 1999).

O modelo de objetos fornece uma moldura para a construção de arquiteturas altamente inteligentes e aplicações distribuídas a nível de transdutores em nós de rede (SCHNEEMAN, 1999). Além disso, fornece acesso padronizado à rede pelos aplicativos do NCAP que definem um modelo de comunicação para a interação intra e inter dispositivos (LEE; SCHNEEMAN, 1996). O acesso padronizado ao transdutor físico é promovido por uma interface programável, baseada em um *driver* do dispositivo, para o modelo da interface. De fato, a aplicação do NCAP está limitada entre dois modelos de *drivers* padronizados do dispositivo. Um é o *driver* de rede e o outro é o *driver* da interface do transdutor, para implementar a interface TII para comunicação do STIM com o NCAP e vice-versa, como mostrado na Figura 2.1. A união consistente de ambos os modelos, possibilita que aplicações sejam reusadas e migradas para outras redes sem maiores esforços de reengenharia.

O modelo de objeto do transdutor em rede, fornece duas interfaces (LEE, 1999; LEE; SCHNEEMAN, 1996):

- **A interface para o bloco transdutor** que encapsula os detalhes de implementação do *hardware* do transdutor com um simples modelo de programação. Este faz a interface do transdutor ver um *driver* de I/O compatível;
- **A interface para o bloco NCAP e portas** que encapsula os detalhes de diferentes implementações de protocolos de rede, por trás de um pequeno conjunto de métodos de comunicação.

Aplicações específicas de comportamento são modelados por blocos de funções. Para produzir o comportamento desejado, o bloco de funções comunica-se com outros blocos. Juntos ligam e desligam o transdutor inteligente (LEE, 1999). Este modelo comum de aplicação de rede independente tem as seguintes vantagens:

- Estabelece um alto grau de interoperabilidade entre transdutores e redes, habilitando assim a capacidade *plug-and-play*;

- Simplificação do suporte de múltiplos protocolos de redes de controle de transdutores.

As classes definidas pelo padrão IEEE 1451.1 consistem de quatro blocos de classes: um bloco físico, um ou mais blocos transdutores, blocos de funções e blocos de rede. Estas classes são ilustradas na Figura 2.21 (JOHNSON, 1997; LEE; SCHNEEMAN, 1996).

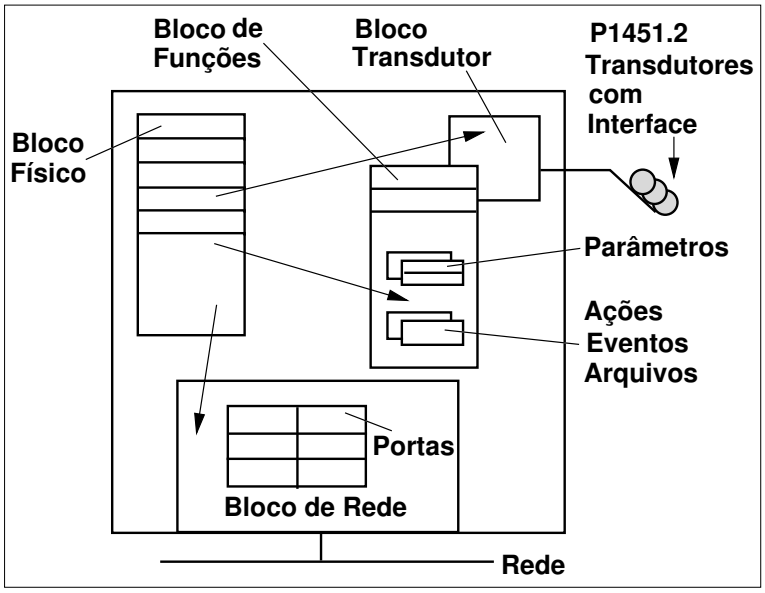

Figura 2.21: Restrições da hierarquia de classes do padrão IEEE 1451.1

Observe como cada bloco de classes pode incluir classes base específicas do modelo. As classes base incluem parâmetros, ações, eventos e arquivos, e executam componentes de classes.

Todas as classes do modelo têm uma classe resumo ou raiz, da qual elas são derivadas. A classe raiz inclui diversos atributos e métodos que são comuns a todas as classes no modelo. Isto fornece a definição de uma classe central a ser usada para instanciação e apagamento. Além disso, também são fornecidos, métodos para receber e moldar atributos dentro de cada classe.

Existem quatro classes objeto encontrada em uma aplicação de sistema: bloco de classes, componentes de classes, serviços de classes, objeto de classes não-IEEE 1451.1. A hierarquia de classes IEEE 1451.1 é mostrada na Figura 2.22 (IEEE, 1999) e pode ser entendida tanto do ponto de vista da indentação ou pela numeração. Por exemplo, a classe Bloco 1.1.1 é uma subclasse da classe Entidade 1.1 que por sua vez é uma subclasse da classe Administrador 1. Assim, classes inferiores na hierarquia de classes da Figura 2.22, herdam atributos, métodos e comportamentos da classe imediatamente superior na hierarquia de classes.

O objetivo do grupo de trabalho NIST/IEEE, que trata dos padrões de interface para transdutores, é utilizar as tecnologias de redes de controle existentes e desenvolver métodos

Administrador 1
Entidade 1.1
Bloco 1.1.1
Bloco NCAP 1.1.1.1
Bloco de Função 1.1.1.2
Base do Bloco Transdutor 1.1.1.3
Bloco Transdutor 1.1.1.3.1
Bloco Transdutor Dot2 1.1.1.3.1.1
Bloco Transdutor Dot3 1.1.1.3.1.2
Bloco Transdutor Dot4 1.1.1.3.1.3
Componente 1.1.2
Parâmetro 1.1.2.1
Parâmetro Com Atualização 1.1.2.1.1
Parâmetro Físico 1.1.2.1.1.1
Parâmetro Escalar 1.1.2.1.1.1.1
Parâmetro Escalar Seriais 1.1.2.1.1.1.1.1
Parâmetro Vetorial 1.1.2.1.1.1.2
Parâmetro Vetorial Seriais 1.1.2.1.1.1.2.1
Parâmetro Temporal 1.1.2.1.2
Ação 1.1.2.2
Arquivo 1.1.2.3
Arquivo Particionado 1.1.2.3.1
Grupo Componente 1.1.2.4
Serviço 1.1.3
Porta Base 1.1.3.1
Porta Cliente Base 1.1.3.1.1
Porta Cliente 1.1.3.1.1.1
Porta Cliente Assincronas 1.1.3.1.1.2
Porta Pública Base 1.1.3.1.2
Porta Pública 1.1.3.1.2.1
Auto Identificando Porta Pública 1.1.3.1.2.2
Evento Gerador de Porta Pública 1.1.3.1.2.2.1
Subscrever Porta 1.1.3.2
Abortar Serviço 1.1.3.3
Condicão de Servico Variável 1.1.3.4

Figura 2.22: Hierarquia de classes do padrão IEEE 1451.1

de conexão padronizados para transdutores inteligentes para redes de controle. Poucas ou nenhuma mudança deve ser requerida para utilização de diferentes métodos de conversão A/D, diferentes microprocessadores ou diferentes protocolos de redes e *transceivers*.

Este objetivo é alcançado por meio da definição de um modelo de objeto comum para os componentes de uma rede de transdutores inteligentes, juntamente com a especificação de interfaces para estes componentes. O modelo de transdutor inteligente em rede é apresentado através de duas visões: visão física e visão lógica.

2.7.1 Visão física

Na primeira visão são ilustrados os componentes físicos do sistema. Esta visão é indicada pelos componentes mostrados com linhas sólidas na Figura 2.23. Nesta Figura é mostrado o modelo composto de sensores e atuadores para formar um transdutor. O transdutor é conectado por meio de uma interface a um microcontrolador ou controlador que está ao lado, interfaceado pela rede. A Especificação da Interface de *Hardware* entre o sensor/atuador e o restante do *hardware* do dispositivo, declarado como o NCAP é

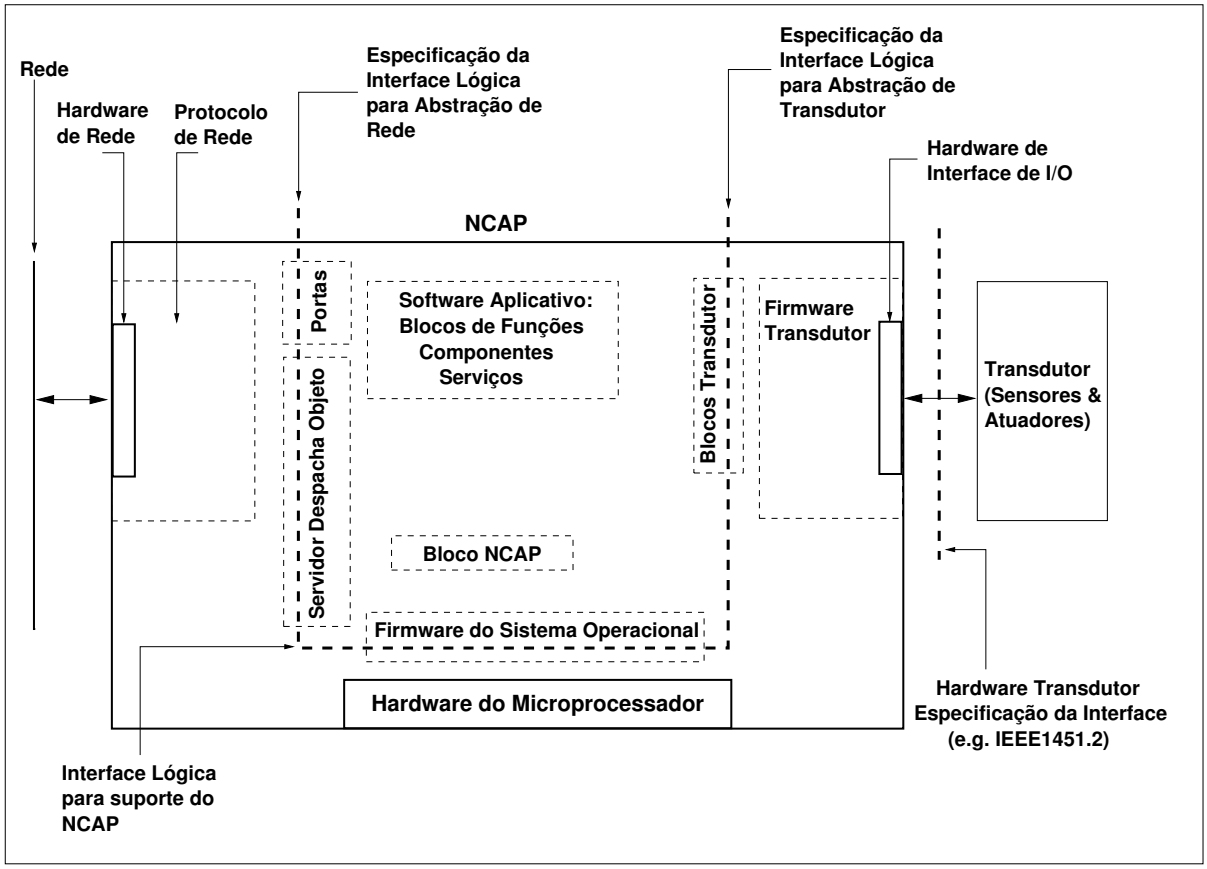

Figura 2.23: Modelo do transdutor inteligente em rede

indicado pela parte da linha tracejada mais espessa do lado direito da Figura 2.23. Essa especificação é mostrada no padrão IEEE 1451.2 (IEEE, 1997).

O *hardware* do NCAP consiste do microcontrolador e de seus circuitos de suporte, bem como *hardware* implementando a camada física da rede embutida e a interface de I/O para o transdutor, como mostrado na Figura 2.23.

2.7.2 Visão lógica

A segunda visão é a visão lógica do sistema e é indicado pelas linhas tracejadas da Figura 2.23. Os componentes lógicos podem ser agrupados em componentes de aplicação e suporte. Os componentes de suporte são o sistema operacional, o protocolo de rede e os componentes do *firmware* do transdutor mostrado na Figura 2.23. O sistema operacional fornece uma interface para aplicações, indicada pela linha tracejada denominada Interface Lógica para suporte do NCAP, no canto inferior esquerdo da Figura 2.23.

A segunda interface lógica, chamada de "especificação da interface lógica para abstração de rede", consiste de portas e componentes "servidor despacha objetos" definidos neste padrão. Esta interface fornece uma abstração para ocultar detalhes de comunicação específicos para uma dada rede dentro de um pequeno conjunto de métodos de

comunicação. Os detalhes desta interface são definidos por este padrão.

A terceira interface lógica, chamada de "especificação da interface lógica para abstração do transdutor", desempenha a mesma função de abstração para os *hardware* e *firmware* específicos para I/O do transdutor. Como conseqüência desta ação, esta interface faz com que os *drivers* de I/O das interfaces dos transdutores sejam vistos. Os detalhes desta interface são definidos por este padrão.

Aplicações são modeladas como blocos de funções em combinação componentes e serviços. O bloco NCAP fornece organização da aplicação e suporte para outros blocos. Todos esses blocos, componentes e serviços são definidos neste padrão. Estas interfaces são opcionais no sentido que nem todas devem ser expostas em uma aplicação.

2.8 Conclusão

Neste capítulo foi apresentada uma síntese detalhada do módulo transdutor inteligente, de acordo com as especificações dos padrões IEEE 1451.2 e IEEE 1451.1. Onde foram abordados os principais conceitos e o princípio de funcionamento do módulo transdutor inteligente segundo estes padrões, de forma a dar ao leitor uma visão global dos padrões e de suas principais partes e como estas partes se interrelacionam para fornecer as funcionalidades do transdutor inteligente, dentro do conjunto de normas e protocolos do IEEE 1451.2 e IEEE 1451.1. Bem como, possibilitar a familiaridade com os mesmos, afim de implementá-los.

Capítulo 3

O hardware usado na implementação

3.1 Introdução

Neste capítulo é apresentado o *hardware* usado na implementação dos padrões IEEE 1451.1 (IEEE, 1999) e IEEE 1451.2 (IEEE, 1997). Uma placa TBM390 (TINI - *Tiny InterNet Interface - Board Model* 390) (DALLAS, 2001), foi usado na implementação do IEEE 1451.1 e o microcontrolador ADuC832 (ANALOG DEVICES, 2002a) na implementação do IEEE 1451.2.

A escolha do microcontrolador ADuC832 e do TINI para as implementações do STIM e do NCAP respectivamente, foi motivada por suas especificações técnicas, adequadas para este fim e pelo baixo custo dos mesmos. Outra motivação para implementação do NCAP foi a inexistência de um NCAP comercial, uma vez que o único NCAP comercial encontrado em 2003 foi o Bfoot-66501 desenvolvido pela HP e que havia saído de circulação, justificando assim a implementação de um novo NCAP. Os detalhes sobre ADuC832 e TINI são mostrados nas subseções seguintes.

Além disso, são abordados aqui os principais aspectos e conceitos da *interface* SPI (ESTL, 2002). A teoria de JNI (*Java Native Interface*) (SUN, 2004), também é abordada, para possibilitar o entendimento da biblioteca spi.tlib (MAXIM/DALLAS, 2001), desenvolvida em *software* para implementar a *interface* SPI do TINI.

3.2 A plataforma TINI

O TINI modelo 390 (DALLAS, 2001) é usado em aplicações embarcadas, tais como, monitoração e controle de dispositivos ou sistemas locais. É ferramenta de desenvolvimento para escrita de servidores *web* embarcados (MAXIM/DALLAS, 2003a). Programado em Java, o TINI é uma referência de projeto (MAXIM/DALLAS, 2003b). O TINI tem um microcontrolador DS80C390 (MAXIM/DALLAS, 2003c) com a JVM (*Java Virtual Machine*)

embarcada. Integrado ao microcontrolador existe os seguintes protocolos de comunicação serial de baixo nível: $RS232-C$, UART (*Universal Asynchronous Receiver Transmitter*), CAN, *1-Wire*. A *interface* SPI, existe implementada em *software* (MAXIM/DALLAS, 2001). O TINI tem controlador *Ethernet*; 512 *kilobytes* de memória Flash ROM; 512 *kilobytes* memória RAM (expandível a até 2 *megabytes*); 2 portas serial integradas e suporte para 2 portas serial externa; 2 controladores CAN integrados; barramentos *1-wire* externo e interno; relógio em tempo real. Com API (*Applications Programming Interface*) JDK 1.1 (*Java Development Kit*) e um SO (Sistema Operacional) com multi-tarefas, multi-*threaded*, pilha TCP/IP (*Transmission Control Protocol / Internet Protocol*) completa, gc (*garbage collection*), porta serial e *drivers 1-Wire*, suporte PPP, um sistema *shell* (Slush) (DALLAS, 2002g) baseado no Unix e com *interface* de acesso TTY, TELNET e FTP (DALLAS, 2002e). Ele também possui I/O TTL de propósito geral com pinos bidirecionais. Seus aplicativos são desenvolvidos em Java, contudo, também é possível desenvolver aplicativos com MN escritos em assembler (SUN, 2004; DALLAS, 2002c, 2002b).

A maioria de suas aplicações, inclusive a descrita neste trabalho, utilizam sua capacidade de rede. Algumas aplicações incluem:

- **Controle industrial** - com o *chip* CAN (*Controller Area Network*) (GMBH, 1991; LAWRENZ, 1997) integrado, servindo como suporte de instrumentação na implementação da automação de equipamentos de fábricas, chaveamento de redes e atuadores.
- **Monitoração e controle de equipamentos via** *web* - pode ser usado para comunicação com equipamentos, para fornecer diagnóstico remoto e coletar dados com o propósito de utilização como dispositivo de monitoração (MAXIM/DALLAS, 2003b).
- **Conversão de protocolos** - usado para conectar dispositivos à rede *Ethernet*.
- **Monitoração de ambiente** - usado como suporte na construção de rede *1-wire* (MAXIM/DALLAS, 2002b), em aplicações que podem enfileirar sensores e informar os resultados das medições a um *host* remoto (MAXIM/DALLAS, 2002d).

Na Figura 3.1 é mostrado o modelo de uso, no qual o TINI é empregado como um conversor de protocolo (ou *link*) entre o dispositivo embarcado e a rede *Ethernet* (MAXIM/DALLAS, 2002f). O dispositivo pode se comunicar com o mundo exterior usando uma porta serial $RS232$ (MAXIM/DALLAS, 2002c), controlador CAN (MAXIM/DALLAS, 2002g, 2003c), rede *1-Wire* (MAXIM/DALLAS, 2002a), *interface* SPI (MAXIM/DALLAS, 2001) ou através de algum tipo de interface paralela. O aplicativo em execução no TINI desempenha a tarefa de comunicar com o dispositivo a ele conectado em sua linguagem nativa, usando um protocolo de comunicação específico para o dispositivo e apresenta o resultado ao sistema remoto, alcançável via rede TCP/IP (*Transmission Control Protocol*

/ Internet Protocol). O *link* fornecido pelo TINI é bi-direcional, permitindo controlar um sistema remotamente, bem como monitorar um dispositivo.

Na Figura 3.1 é focalizada um sistema embarcado que controla e fornece conectividade de rede a um dispositivo. Entretanto, o TINI também interconecta vários tipos de redes fazendo uma ponte entre uma rede menor composta por dispositivos leves e baratos e grandes redes mundiais TCP/IP, como a *internet*.

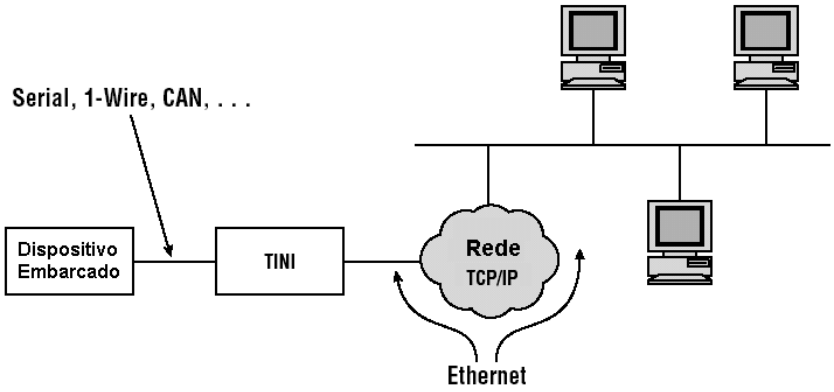

Figura 3.1: Conversão de protocolo

Sistemas baseados no TINI freqüentemente fornecem um *display* remoto através da implementação de um servidor de rede, tal como servidor HTTP (*HiperText Transfer Protocol*), permitindo ao usuário interagir com o sistema utilizando uma rede cliente, tal como uma *Web browser*. *Displays* locais e entrada de dados podem ser obtidos pelo interfaceamento de um PDA (*Personal Digital Assistant*) (MAXIM/DALLAS, 2002e) sobre um *link* sem fio, tal como um infra-vermelho ou um *link* serial por cabo físico.

3.2.1 Mapa de memória

O mapa de memória especifica a memória e onde outros dispositivos periféricos são mapeados nos espaços de endereçamento do microcontrolador. O mapa de memória usado pelo TINI com o tamanho máximo dos segmentos é mostrado na Figura 3.2 e consiste dos seguintes seguimentos:

- Código;
- Dados;
- Periféricos.

O inicio do endereço dos diferentes segmentos são sempre os mesmos, mas o endereço final pode ser menor que o indicado, dependendo da quantidade de memória ocupada. O requisito mínimo de memória para códigos e dados é de 512 *kilobytes* cada.

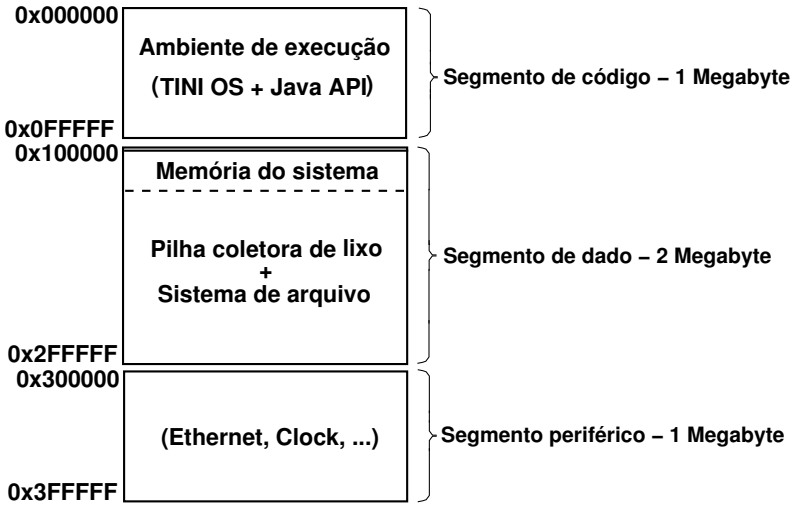

Figura 3.2: Mapa de memória do TINI

Os segmentos de códigos e dados são ocupados pela memória do *chip* e o segmento periférico é ocupado por outros tipos de componentes de *hardware* tais como controlador *Ethernet* e *clock* em tempo real. Outros dispositivos periféricos que suportam um barramento de interface paralela compatível com o barramento do microcontrolador também pode ser mapeado pelo periférico. Deve-se ter cautela ao adicionar *hardware*, pois dessa forma também adiciona-se carregamento capacitivo, para um dos dois ou ambos endereços de barramento ou de dados (dependendo do dispositivo). O projetista do sistema deve está consciente deste carregamento capacitivo para assegurar a realização operacional do sistema (DALLAS, 2001).

I/Os integrados

Os dispositivos periféricos descritos nas seções anteriores são todos interfaceados pelo barramento de dados e endereços do microcontrolador. Entretanto, uma ampla faixa de dispositivos que estão interagindo com o TINI para habilitar a rede não tem suporte completo ao barramento paralelo. Isto usualmente resulta na diminuição da largura de faixa de comunicação. Mas a interface serial também reduz a quantidade de pinos requisitados, simplificam a comunicação e freqüentemente tem custo mais baixo quando comparado com dispositivos que tem algum tipo de barramento de interface paralela. Interrupções seriais também têm a vantagem de não carregar os barramentos do microcontrolador. Integrado ao microcontrolador existe os seguintes protocolos de comunicação serial de baixo nível:

- **Comunicação serial** - protocolos seriais síncronos, com interface *2-wire* e comunicação serial síncrona, baseada no padrão $RS232 - C$. O controlador do TINI tem dois circuitos integrados UART para facilitar a comunicação serial;

- **CAN** - originalmente desenvolvido pela *Bosch-Siemens*, CAN é descrito em dois padrões ISO. ISO11898 para aplicações de alta velocidade e ISO11519-2 para aplicações de baixa velocidade. Ele fornece barramento de comunicação serial, comumente usada na indústria automotiva e em aplicações de controle industrial. O microcontrolador do TINI tem dois controladores CAN integrados;

- **Rede** *1-Wire* - desenvolvida pela *Dallas Semiconductor* é uma rede de pequenos sensores, atuadores e células de memória. Todos os dispositivos compartilham o mesmo condutor para comunicação e energização.

- **TTL I/O** - de propósito geral, o microcontrolador tem portas com pinos bidirecionais que podem ser usados em diversas tarefas de controle e não estão necessariamente presos a qualquer tipo de dispositivo de comunicação serial.

Utilizando a capacidade integrada de I/Os do microcontrolador, ao invés de I/Os mapeados na memória, reduz-se, o número total de dispositivos e o custo de comunicação com um dispositivo externo, pois ele reduz a comunicação da CPU com o dispositivo interfaceado pelo barramento do microcontrolador. Por exemplo, a CPU do microcontrolador roda em plena velocidade, executando o ambiente, enquanto, simultaneamente o UART envia e recebe caracteres serialmente. Por outro lado, a comunicação com o periférico interfaceado pelo barramento, requer que a CPU pare quando ele está em operação e quando ele executa instruções para ler dados do dispositivo ou escrever dados para o dispositivo.

3.3 Ambiente de desenvolvimento

O termo plataforma de desenvolvimento refere-se ao computador utilizado para criar, construir e carregar aplicativos no TINI. Esta máquina deve possuir o ambiente de desenvolvimento Java e o ambiente de execução. Além disso, o TINI deve estar conectado à rede *Ethernet* e/ou ao PC via porta serial.

Desde que todos as ferramentas requisitadas sejam escritas em Java, aplicativos para o TINI podem ser desenvolvidos sob qualquer um dos seguintes SO: qualquer win32 (como Windows 95,98,NT,2000,XP), Linux e Solaris.

Além de um dos SO citados e da porta serial, a plataforma de desenvolvimento deve ter os seguintes softwares corretamente instalados:

- Ambiente de desenvolvimento Java;

- API de comunicação Java;

- Kit de desenvolvimento de *Software* TINI. O *download* deste *software* é feito do site http:/www.ibutton.com/TINI/software/index.html.

De posse desses *software* deve-se ler atentamente o arquivo README.txt (DALLAS, 2002e) que o acompanha (DALLAS, 2001), no qual é descrito como e onde setar diversas variáveis e instalar aplicativos do sistema. Além de fazer uma explanação sucinta dos recursos do TINI, direcionando os detalhes da discussão para diversos outros arquivos.txt da documentação.

A instalação do SO (DALLAS, 2003c) é feita, em duas etapas, após a instalação dos aplicativos e setagem das variáveis (DALLAS, 2002e), com cabo direto, via porta serial (DALLAS, 2002h) conectada ao TINI e ao computador de desenvolvimento (DALLAS, 2001).

A primeira etapa começa digitando em um terminal (*prompt* do MSDOS, no caso de estar usando o Windows, como SO) o comando *JavaKit* (DALLAS, 2002f), abra-se uma interface. No menu *File* dessa interface deve-se selecionar o botão *Load File* e selecionar os arquivos tini.tbin e slush.tbin, os quais estão localizados no diretório <Diretório de instalação do TINI $>$ $\backslash bin\backslash$. Finalizada essa fase digita-se $b18$, para selecionar as últimas 64K posições de memória da pilha do TINI. Em seguida, digita-se $f0$ para apagar o conteúdo das últimas 64K posições de memória da pilha do TINI, forçando o SO inicializar a pilha, o sistema de arquivos e todos os outros ambientes persistentes. Finalmente digita-se E de $EXIT$ para ter acesso ao campo para digitação de *login* e senha de acesso ao TINI [1] (DALLAS, 2002e, 2003c, 2001). O TINI já pode ser configurado na Rede (DALLAS, 2001).

A segunda etapa da instalação do SO começa carregando no TINI, por FTP, os arquivos LoaderLoader.tini, loaderloader.tlib, tini111-bank0.tbin2. Execute no *prompt* do *slush* (DALLAS, 2002g) (*prompt* do TINI) o comando java LoaderLoader.tini tini111-bank0.tbin2. Nessa fase recomenda-se muita cautela ao executar esse último comando, pois caso ocorra interrupção no fornecimento de energia ao TINI, o conteúdo de sua BIOS é violado, forçando o usuário enviá-lo ao fabricante para regravação do conteúdo da mesma (DALLAS, 2003c).

Além de todas as potencialidades já destacadas nesse documento, com o TINI ainda é possível fazer aplicações usando código nativos (DALLAS, 2002b, 2002c, 2002d) escritos em assembler. Entretanto, o sistema também tem limitações tais como: a menor taxa de carregamento do sistema é de 19900 *bauds*, módulos nativos não podem ser superior a 64 *kilobyte* (mais detalhes sobre limitações em (DALLAS, 2002a)).

[1] Exite dois *logins default*: o de *root* e de *guest*, com senhas tini e *guest*, respectivamente

3.4 Ambiente de execução do TINI

Fornecer *hardware* essencial para desenvolvimento de dispositivos de rede embarcada é apenas metade do trabalho. Uma grande quantidade de *software* também é requisitada para o desenvolvimento de aplicações, com as quais, deve-se ter preocupação a respeito de detalhes de camadas crescentes da infraestrutura que fornece suporte à execução de múltiplas tarefas, pilhas de protocolos de rede e aplicações de interfaces programáveis. Um ambiente de execução bem definido que proporcione todas essas características permite desenvolver o foco primário sobre os detalhes da aplicação. Por essas razões um ambiente de execução foi desenvolvido desde o início, como parte integrante da plataforma global.

O *software* engloba o ambiente de execução do TINI, o qual pode ser dividido em duas categorias: o código nativo executado diretamente pelo microcontrolador e a API interpretado como código de *bytes bytecodes* pelo JVM. O código dos aplicativos é escrito em Java e utilizam a API (DALLAS, 2002b) para explorar a capacidade do ambiente de execução nativo e os recursos de *hardware* subjacentes. Também é possível escrever bibliotecas que podem ser carregadas de dentro de um aplicativo que apresente requisitos estritos de tempo real. Uma representação gráfica do ambiente de execução é mostrada na Figura 3.3.

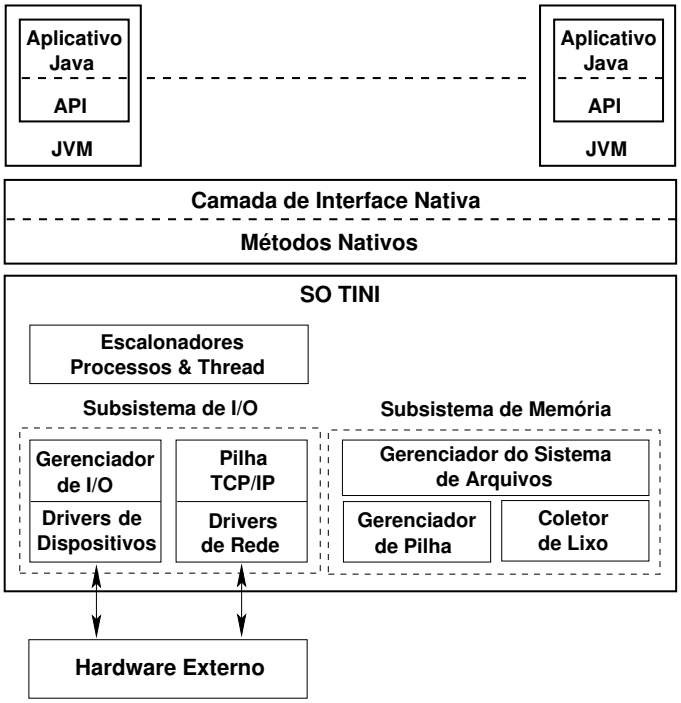

Figura 3.3: Ambiente de execução do TINI

No TINI, os aplicativos Java têm permissões para acessar todos os recursos do sistema. Isto, é particularmente importante para aplicações embarcadas, uma vez que são quase acopladas com os dispositivos físicos. Também, diferentemente de outras plataformas

Java, no TINI não há sistema administrador que desempenhe configuração e manutenção. Isto significa que os aplicativos são responsáveis pela configuração, bem como pelo controle do sistema inteiro. Por essas razões, um aplicativo que controla um sistema embarcado deve ter acesso total a todas as funcionalidades de baixo-nível do SO.

3.4.1 Visão geral da API

A versão da API do ambiente de execução combina classes de muitos pacotes definidos na *Sun* pelo *kit* de desenvolvedores Java (*Java Developer's Kit - JDK*) versão 1.1.8 com classes específicas para o TINI que expõem as capacidades do sistema. As classes específicas do TINI são todas definidas como sub-pacotes do pacote global *com.dalsemi*. As classes que são incluídas no ambiente de execução são conhecidas como a porção embutida da API. Também existem outras classes definidas na API do TINI que podem ser incluídas em aplicações durante o processo de construção das mesmas.

A API inclui implementações de muitas classes nos seguintes pacotes do núcleo Java: *java.lang*, *java.io*, *Java.net*, *Java.util*.

Os pacotes *com.dalsemi* são:

- *com.dalsemi.system* - as classes neste pacote fornecem acesso a muitas formas integradas de I/Os, incluindo a porta serial síncrona *2-wire*, o barramento de dados do microcontrolador e pinos de portas individuais. Ele também contém classes para configuração de recursos do sistema tais como *clock*, cão de guarda temporizado (*watchdog timer*) e interrupção externa;

- *com.dalsemi.tininet* - este pacote contém uma classe chamada $TININet$ que fornece métodos estáticos para enfileirar e setar diversos parâmetros de rede do sistema, tais como, endereço IP e máscara de sub-rede. Sub-pacotes de *com.dalsemi.tininet* fornecem suporte a protocolos de redes tais como DHCP (*Dynamic Host Configuration Protocol*), ICMP (*Internet Control Message Protocol*) e DNS (*Domain Name System*);

- *com.dalsemi.shell* - classes neste pacote e seus sub-pacotes implementam a infraestrutura para aplicações de comandos *shell*. Servidores de TELNET e FTP são implementados em classes contidas em sub-pacotes de *com.dalsemi.Shell*. Estes servidores também podem ser usados por outros aplicativos, enquanto comandos *shells* fornecem acesso a aplicativos clientes por TELNET e FTP;

- *com.dalsemi.comm* - este pacote contém classes de baixo-nível para acesso aos controladores CAN. Ele também contém diversas classes para configuração e comunicação com as portas seriais do sistema. Entretanto, estas classes são livremente

usadas por outras aplicações. Acesso à porta serial é promovido pela implementação da API de comunicação Java da *Sun*, os quais são definidos no pacote *javax.com*;

- *com.dalsemi.onewire* - este é o gerente (*root*) da hierarquia de pacotes para a API *1-wire*. Diferentemente dos outros pacotes listados acima, a API *1-wire* também é suportada por outras plataformas em Java. O pacote *com.dalsemi.onewire.container* fornece classes conhecidas como recipientes (*containers*), que compreendem o comportamento específico de *chips 1-wire*. Para avaliar o consumo preciso de espaço na memória *flash*, dispositivos específicos contêm classes que não são incluídas na construção como parte da aplicação.

3.4.2 A Java Virtual Machine (JVM)

A quantidade mínima de memória requerida do TINI pelo JVM é inferior a 40 *kilobytes*. A despeito de seu tamanho, ele suporta muitas das funcionalidades fornecidas pela implementação total da JVM, incluindo as seguintes:

- Suporte total para *threads* (SUN, 2004);
- Suporte para todos os tipos de primitivas (STALLINGS, 1999);
- *Strings*.

Entretanto, também existe importantes omissões, tais como:

- Carregamento dinâmico de classes;
- Finalização[2] de objetos.

Declarar que a JVM não suporta carregamento dinâmico de arquivos de classe pode causar a impressão que os métodos $Class.forName$ e $Class.newInstance$ não são suportados. De fato, ambos são implementados junto com diversos outros métodos definidos na classe $Class$. Muitas das classes na API contam com esta capacidade para diversas tarefas, incluindo criação e conversão de caractere em *byte* e carregamento das restrições do *chip 1-Wire*. Entretanto, se um *thread* de execução invoca $forName$ e o passa a um *string* especificando a classe que não existe na API construída ou no aplicativo corrente em execução, $forName$ apresentará uma $ClassNotFoundException$ depois que carregar a imagem binária da classe especificada do aplicativo corrente.

[2] Um método que finaliza um objeto pode ser explicitamente invocado por um *thread* Java de execução, mas não é automaticamente executado, antes ele é recuperado pelo *garbage collector*.

O carregamento de classes é efetivamente explicitado em duas fases. A primeira é executada pelo utilitário *TINIConvertor* (DALLAS, 2002e, 2003a, 2003b) sobre uma máquina *host* de desenvolvimento. O conversor atua na definição completa da união constante de todas as classes usadas pelo aplicativo. Classes do aplicativo pode referenciar métodos e campos em outras classes do aplicativo ou da API embutido. O resultado deste processo de conversão é uma imagem binária que pode ser executada diretamente pela JVM do TINI. Qualquer inconsistência não resolvida, resulta na abortagem direta do conversor, antes da geração da imagem executável. A segunda fase do processo de carregamento de classes, executa os métodos inicializados pelas classes no TINI. Quando um novo aplicativo é executado, todas as classes que inicializam métodos são executadas pelas classes da API, seguido por todos os métodos inicializados pelas classes do aplicativo. O efeito cascata deste modelo de carregamento explícito de classes existente no aplicativo, por *default*, carrega todas as classes definidas na API, bem como, as classes especificadas pelo aplicativo. Isto não aumenta a quantidade de memória requisitada pela imagem binária do aplicativo, pois as imagens convertidas das classes da API são armazenadas separadamente na memória *flash* como parte do ambiente de execução. Existem omissões a cerca do procedimento funcional, também existe limitações de recursos, tais como, um máximo de 16 *threads* ativos executando (DALLAS, 2002a).

3.4.3 Métodos Nativos

A camada nativa (DALLAS, 2002c) mostrada na Figura 3.3 representa a coleção de métodos nativos (MN) (DALLAS, 2002b) que suportam a API exposta pela infra-estrutura fornecida pelo SO do TINI. Isso inclui suporte a acesso completo a camadas do protocolo de rede, bem como *drivers* para dispositivos fora da rede. Ele também inclui métodos para configuração e acesso a recursos do sistema tais como cão de guarda temporizado e *clock* em tempo real.

Entre a atual implementação do MN e o interpretador de código Java existe uma camada conhecida como interface de MN. A interface de MN é o limite que deve ser atingido para chavear o contexto de execução entre o código que está sendo executado pela JVM e o MN. A interface de MN do TINI (*TINI's native method interface - TNI*) fornece um mecanismo bastante leve para atingir esse limite. Seu análogo para outras plataformas Java é a JNI (*Java Native Interface - JNI*). A invocação de MN pelo ambiente de execução provoca uma redução máxima do contexto do chaveamento de *overhead*.

Aplicações que requisitem adaptações para MN podem ser fornecidas por bibliotecas que podem ser carregadas no sistema em execução usando o método *loadLibrary*, definido na classe *Java.lang.Runtime*

public static void loadLibrary(string libname)

onde o parâmetro *libname* especifica o nome do arquivo biblioteca nativa (mais detalhes em métodos_nativos.txt (DALLAS, 2002c), native_API.txt (DALLAS, 2002b) e na seção 3.5). Nestas referências é descrito em detalhes o processo para escrever e construir bibliotecas nativas.

3.4.4 O SO do TINI

O SO do TINI está abaixo da camada do ambiente de execução. Ele é responsável pelo gerenciamento de todos os recursos do sistema, incluindo acesso a memória, escalonamento de múltiplos processos e execução de *threads* (SUN, 2004), e interage com componentes de *hardware* internos e externos. Apesar do SO ser um corpo de código complexo que desempenha muitas tarefas independentes, ele é razoavelmente bem representado pela soma dos três seguintes elementos majoritários:

- Escalonamento de processos e *threads*;
- Subsistema de gerenciamento de memória;
- Subsistema de gerenciamento de I/Os.

Os escalonadores - o SO contém ambos os processos e módulos escalonadores de *threads* que guiam a execução de código a nível de aplicativo. Os escalonadores são executados por um dos *timers* do microcontrolador que gera uma interrupção de alta prioridade a cada milissegundo. As rotinas do serviço de interrupção do *timer* (*interrupt service routine-ISR*) também executam ou inicializam as seguintes tarefas:

- Atualiza o contador de tempo do sistema a cada milissegundo;
- Executa o escalonador de *threads* a cada 2 milissegundos;
- Executa os módulos de *drivers* dos dispositivos a cada 4 milissegundos;
- Executa o escalonador de processos a cada 8 milissegundos.

Os processos são escalonados pelo escalonamento *round robin* (LIU, 2000). Cada processo é executado em intervalos de tempo de 8 milissegundos. Após expirar o tempo do intervalo, o processo é enviado para o fim de uma fila de processos ativos que esperam seu retorno para execução de outro intervalo de tempo. Quando existem múltiplos processos no sistema, um único processo pode utilizar quase todos os recursos, desde que, exista apenas um processo ativo competindo pelo tempo de execução. Cada processo tem seu próprio escalonador de *threads* operando independentemente. Os *threads* a nível nativo são corporativos, cada *thread* voluntariamente cede o controle da CPU. Entretanto, para

o caso de aplicativos Java, *thread* são preemptados devido a JVM assegurar que cada *thread* cede a CPU depois de 2 milissegundos. *Threads* também são escalonados pelo escalonamento *round-robin*.

O subsistema de gerenciamento de memória - desempenha as seguintes tarefas:

- Aloca memória da pilha para ambos os processos do sistema e Java;
- O gc (*garbage collector*) é gerado automaticamente pelos processos Java;
- Gerência de arquivos do sistema.

Na Figura 3.2 é mostrado o segmento de dados que contém toda memória rápida de leitura escrita usada pelo ambiente de execução. A porção de memória da área do sistema para o fim do segmento de dados é chamada pilha. A pilha representa a capacidade de memória do sistema. O acesso à pilha é controlado por um conjunto central de rotinas de alocação de memória. A operação básica dessas rotinas é muito similar à operação *malloc* de C. Uma exceção ocorre quando muitas operações de alocação de memória limpam todos os *bytes* de um bloco de memória alocado para 0 antes de retornar o bloco para o chamador. Muitos blocos de memória são alocados da pilha em benefício de novas operações executadas pela JVM ou por operações do sistema de arquivos. Raramente existem blocos de memória livres explicitamente. Isto é verdade devido a maioria da memória ser consumida pelas tarefas do sistema e de toda memória consumida pelo JVM em suporte a aplicativos Java. Memória é liberada pelo gc que fica executando como um processo separado do sistema. O processo gc é criado quando o sistema é ligado. Ele existe apenas em processos não-Java que são sempre criados. Sujeito normalmente à condicional de memória, o processo gc passa a maioria do tempo em estado inativo. Quando o gc, ou qualquer outro processo, estiver inativo ele não consome tempo de processamento. Ele é executado em uma das três maneiras:

- Um aplicativo invoca explicitamente o método gc definido na classe *Java.lang.System*;
- Uma nova operação reduz a quantidade de memória a níveis abaixo de 64 *kilobytes*;
- Um processo Java termina.

Quando o gc executa, ele não limpa o lixo da pilha inteira. Ele limpa apenas o lixo criado pelo processo que o executou. Quando o processo termina, toda memória consumida pelo processo, incluindo aquela mantida pelos objetos e estrutura interna da JVM é liberada.

Subsistema de I/Os - Os subsistemas de I/Os são divididos em duas categorias: I/Os em rede e I/Os não em rede. I/Os em rede, aqui referidos, são estritamente os I/Os

para grandes redes mundiais TCP/IP. Redes CAN e *1-wire*, embora sejam tecnologia de rede não são caracterizadas como I/Os em redes.

O TCP/IP e o gerenciador de I/Os são implementados como processos *kernel* independentes. Estes processos são guiados por um sistema temporizado de 4 milissegundos. O gerenciador de I/Os controla todos os dispositivos não em rede. Pedidos de I/Os gerados por códigos de aplicativos passam todos pelo gerenciador de I/Os que os direcionam apropriadamente. Certos pedidos de I/Os são enviados diretamente ao *hardware* do dispositivo conectado. Por exemplo, não existem *drivers* embutidos para comunicação com dispositivos arbitrários conectados à expansão do barramento paralelo. Neste caso, o aplicativo Java é responsável pelo gerenciamento de todos os detalhes de baixo nível da comunicação com o dispositivo.

O protocolo TCP/IP é um dos maiores blocos de código nativo do ambiente de execução. Ele fornece muitos recursos de rede encontrados em plataformas maiores e é suficientemente rico em funcionalidades para suportar a implementação total do pacote *Java.net*. O protocolo suporta múltiplas interfaces de rede, incluindo *Ethernet*, para LANs (*Local Area Networking*) de alta velocidade e PPP (*Point-to-Point Protocol*) sobre um *link* serial para rede *dial-up* remota usando um modem analógico. A interface *Ethernet* é gerenciada por um *driver* separado que desempenha toda comunicação com o controlador *Ethernet*. O PPP é um pouco diferente, ele atualmente conta com um *driver* para porta serial de baixo nível para entregar mensagens à rede pela porta física de comunicação.

3.5 Conceitos da JNI

A JNI (SUN, 2004) é uma interface de programação nativa para Java que é parte do JDK. Programas escritos usando a JNI, têm a garantia de seu código ser completamente portátil sobre todas as plataformas. A JNI possibilita que códigos escritos em Java, executados pela JVM operem com aplicativos e bibliotecas escritas em outras linguagens, tais como C, C++ e assembler. Além disso a invocação da API permite embarcar a JVM em aplicações nativas.

Programadores usam a JNI para escrever MN (DALLAS, 2002c) para tratar situações em que um aplicativo não possa ser escrito inteiramente em Java. MN e a JNI podem ser usados nas seguintes situações (SUN, 2004):

- a biblioteca de classes padrão Java, não suporta as características da plataforma dependente, necessária para a aplicação;
- o usuário tem uma biblioteca ou aplicativo escrito em outra linguagem de progra-

mação e deseja torná-la acessível por aplicativos Java;

- ao desejar implementar um pequeno trecho de código com tempo crítico em uma linguagem de programação de baixo nível, como assembler, tendo seu aplicativo em Java para chamar estas funções.

A programação através de quadros JNI possibilita que MN sejam usados em muitas operações. Assim, os MNs devem representar o legado de aplicações ou devem ser explicitamente escritos para resolver problemas, que são melhor tratados fora do ambiente de programação Java. MNs podem facilmente chamar métodos Java. O MN, usando o quadro JNI, pode chamar o método java existente, passar a ele os parâmetros requisitados e pegar os resultados retornados, assim que o método for executado.

A JNI habilita o usuário a usar as vantagens da linguagem de programação Java do seu MN. Em particular, o usuário pode pegar e lançar exceções do MN e ter estas exceções tratadas no aplicativo Java. MNs também podem pegar informações sobre classes Java. Chamando funções JNI especiais, MNs podem carregar classes Java e obter informações de classe. Finalmente, MNs pode usar a JNI para desempenhar tipos de checagem de tempo de execução. Por exemplo, na Figura 3.4 é mostrado como um programa em C pode usar a JNI para conectar bibliotecas Java, chamar métodos Java, usar classes Java e assim por diante.

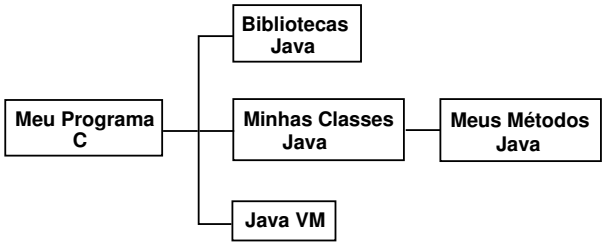

Figura 3.4: Exemplo do uso da JNI para conectar bibliotecas Java, chamar métodos Java e usar classes Java

Na Figura 3.5 é ilustrada a chamada de funções em linguagem nativa de um aplicativo Java. Neste diagrama são mostradas as muitas possibilidades de como utilizar a JNI de um programa Java, incluindo chamadas de rotinas em C, usando classes C++, chamadas de rotinas em assembler e assim por diante.

É visível que a JNI funciona como uma espécie de "junção" entre Java e aplicações nativas. No diagrama da Figura 3.6 é mostrado como a JNI fixa o lado assembler de um aplicativo ao lado Java.

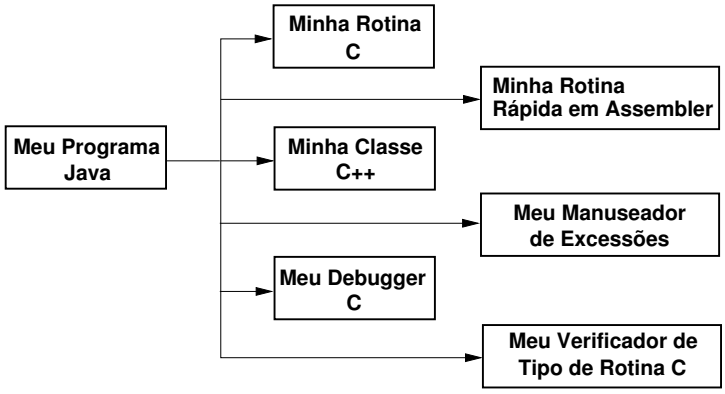

Figura 3.5: Diagrama mostrando as possibilidades de usar a JNI de programa Java para a chamada de rotinas

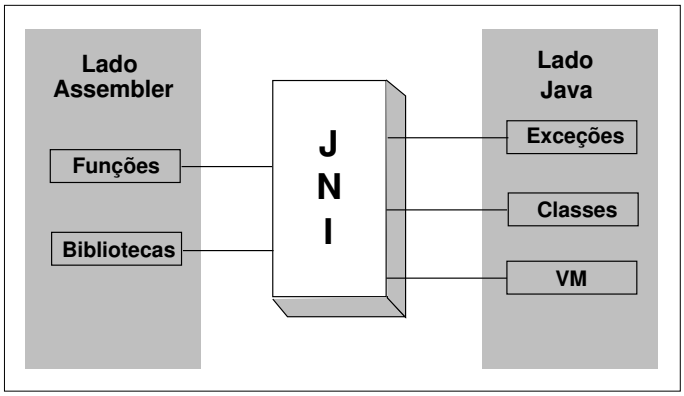

Figura 3.6: Conexão JNI do lado C de um aplicativo ao lado Java

3.5.1 Escrevendo um aplicativo com MNs

Escrever MNs para um aplicativo Java é um processo que segue os seguintes passos (SUN, 2004; DALLAS, 2002c, 2002b):

1. Escreve-se o programa em Java, criando a classe Java que declara o MN. Esta classe contém a declaração ou assinatura para o MN. Ela também inclui o método *main* que chama o MN;

2. Compila-se a classe Java que declara o MN e o método *main*;

3. Escreve-se a implementação do MN na linguagem de programação nativa. No caso do TINI, essa linguagem é assembler;

4. Compila-se os arquivos cabeçalho e a implementação em um arquivo biblioteca compartilhado. Para o TINI, corresponde a assemblar o código do MN segundo sua API;

5. Carrega-se a biblioteca nativa no aplicativo desenvolvido. Para o TINI, isso é feito

usando o *TINIConvertor*, com o comando -n <NomeDaBiblioteca>" para especificar a localização da biblioteca nativa;

6. Usa-se *loadLibrary* do aplicativo Java para carregar a biblioteca nativa;

7. Utiliza-se FTP para carregar e TELNET para executar no *hardware*, o arquivo.TINI do aplicativo, gerado nesse processo.

A representação seqüencial dos passos usados para escrever aplicativos com bibliotecas nativas atachadas são mostrados na Figura 3.7.

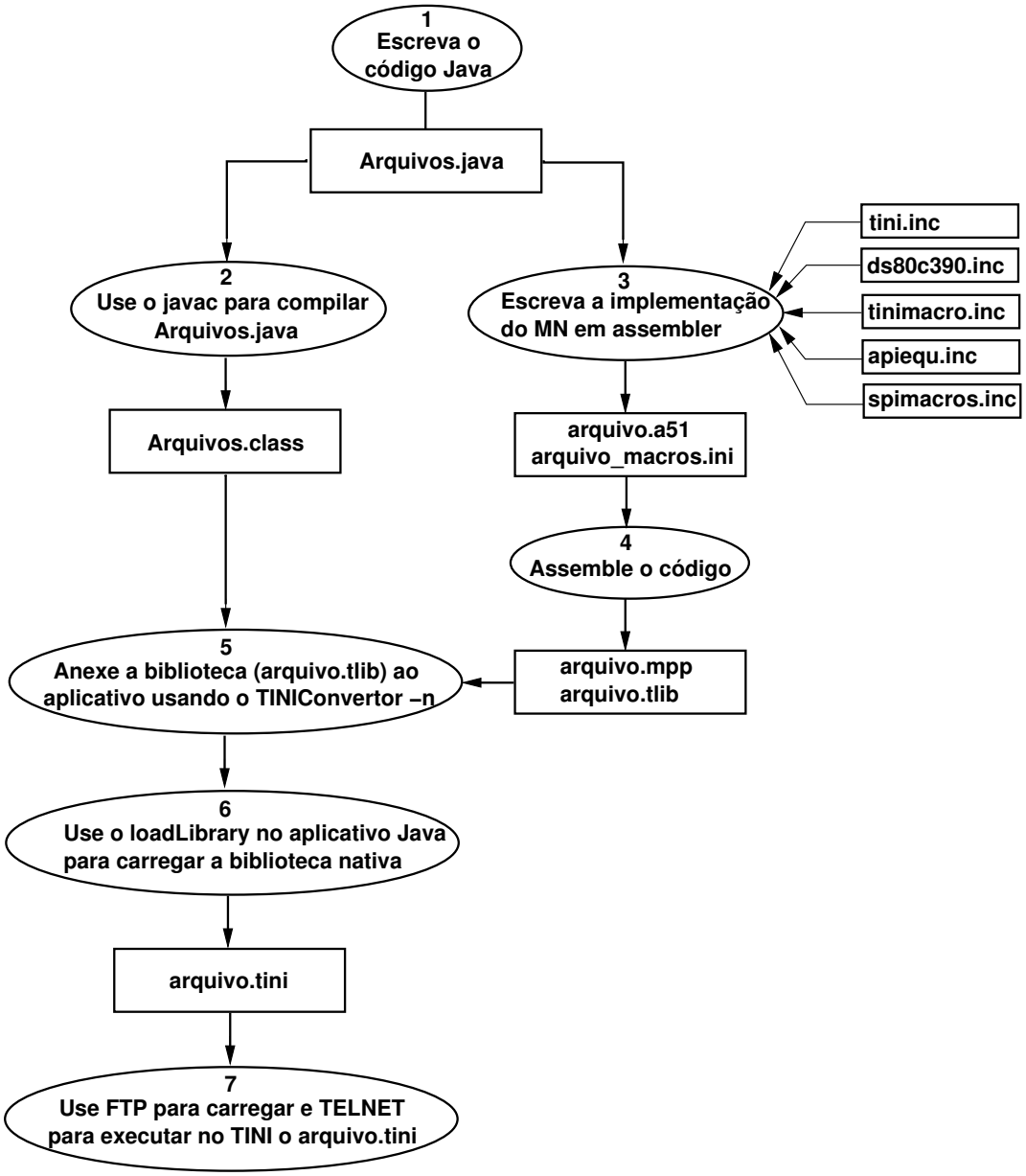

Figura 3.7: Seqüência de passos para construção de aplicativo com biblioteca nativa.

3.6 O microcontrolador ADuC832

O ADuC832 (ANALOG DEVICES, 2002a) é um microcontrolador com interface de usuário integrado em um único *chip*. Baseado no 80C52 da Intel, o ADuC832 possui os seguintes recursos: 1 MCU (*Microcontroller Unit*) de 8 *bits*; 2 conversores DA de 12 *bits*; 1 conversor AD de 12 *bits* do tipo aproximação sucessiva com 8 canais multiplexados com auto-calibração; 2 saídas PWMs de 16 *bits*; 4 portas de I/O serial (SPI, I^2C e UART); 4 *kBytes* de memória *Flash/EE* não-volátil para dados; 256 *bytes* de memória RAM; 2 *kBytes* de memória RAM estendida; 62 *kBytes* de memória *Flash/EE* para memória de programa. Os aplicativos desenvolvidos para o ADuC832 são carregados e executados no *hardware* via porta serial com o WSD (*Window Serial Donwloader*), sem o auxílio de circuitos externos, caracterizando assim a programação ISP (*In-System Programming*). O ADuC832 pode ser programado em linguagem C, utilizando o compilador *Keil uVision* (KEIL SOFTWARE, 2001). O ADuC832 suporta os sistemas de desenvolvimento *QuickStart* e *QuickStart Plus*, os quais são ferramentas de desenvolvimento de *hardware* e *software* de baixo custo (ANALOG DEVICES, 2002a).

O dispositivo opera a partir de um cristal de 32 kHz sobre um PLL, interno ao *chip*, gerando uma alta freqüência de *clock* de 16,77 MHz. Este *clock*, está em conformidade com um roteador através do qual um divisor de *clock* programável gera a freqüência de operação do *clock* no núcleo da MCU. Seu conjunto de instruções é compatível com o conjunto de instruções do processador 8051 com ciclo de máquina de 12 períodos de *clock*. Tem 12 fontes de interrupção, com 2 níveis de prioridade, 2 ponteiros de dados e um apontador de pilha estendido de 11 *bits*, cão de guarda, contador de intervalo de tempo, 3 contadores/*timers*. O *timer* 3 é para geração de taxa de comunicação. O diagrama de blocos funcional detalhado do ADuC832 é mostrado na Figura 3.8.

O ADuC832 opera com alimentação de tensão de 3 a 5 *volts* DC e solicita uma corrente de operação mínima de 25 μA em 3 *volts* DC. É usado em aplicações de redes ópticas - controle de potência de *laser*, sistemas de estação base, instrumentação de precisão (sensores inteligentes), sistemas de capturas de transitórios, sistema de aquisição de dados e sistemas de comunicações.

3.6.1 Escrevendo um aplicativo para o ADuC832

A construção e execução de um aplicativo para o ADuC832 é uma rotina que segue os seguintes passos (ANALOG DEVICES, 2001):

1. Cria-se um novo projeto usando o ambiente de desenvolvimento *Keil uVision*. Nesse passo são gerados os arquivos: arquivo.uv2 e arquivo.plg;

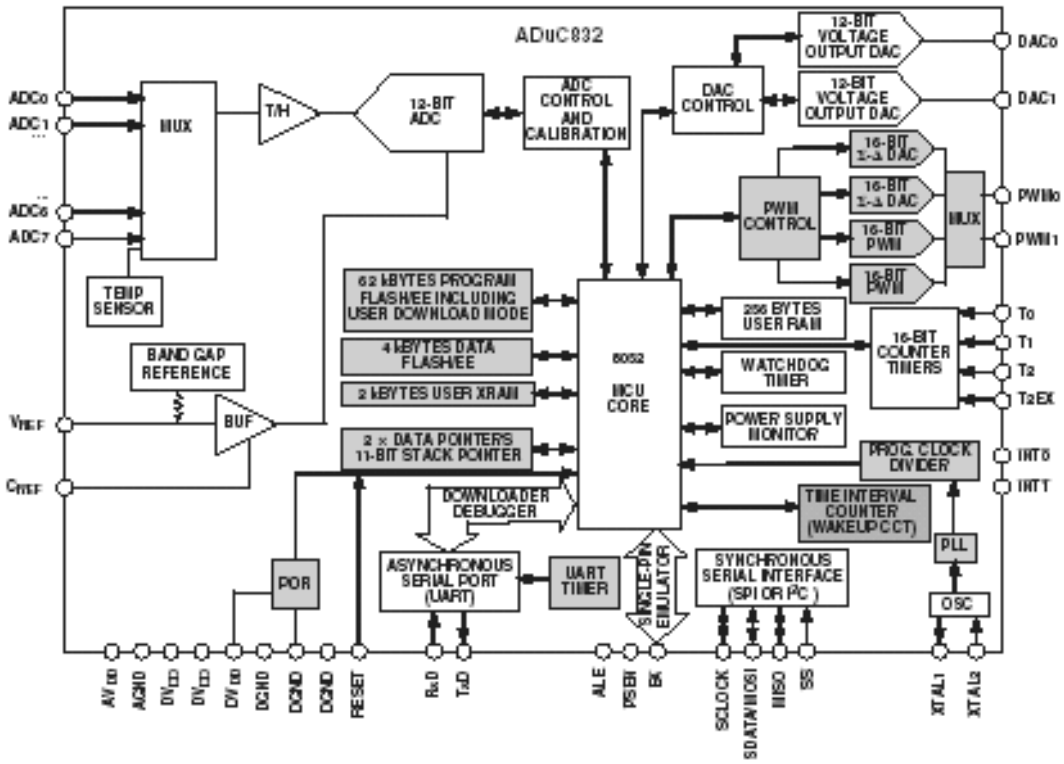

Figura 3.8: Diagrama de blocos do ADuC832

2. Escreve-se o código C do aplicativo. Isso pode ser feito no ambiente de desenvolvimento *Keil uVision*;

3. Utiliza-se o ambiente de desenvolvimento *Keil uVision* (KEIL SOFTWARE, 2001) para compilar e depurar os arquivos.c do aplicativo escrito. Se o aplicativo for compilado sem erro, os seguintes arquivos são gerados: arquivo.hex, arquivo.m51, arquivo.lst, arquivo.obj, arquivo.Opt, arquivo.Inp;

4. Utiliza-se algum *software* de *download* serial (foi usado o WSD) para carregar e executar o arquivo.hex no ADuC832.

A representação seqüencial dos passos usados para escrever aplicativos para o ADuC832 é ilustrado na Figura 3.9.

3.7 A interface SPI

A interface SPI é uma porta de entrada e saída serial síncrona com velocidade da ordem de alguns *megabits* por segundo e permite que dados sejam sincronamente transmitidos e recebidos simultaneamente, isto é, *full duplex* (TEXAS INSTRUMENTS, 2003; ANALOG DEVICES, 2002a; ESTL, 2002; MARTINEZ, 1997; MOTOROLA, 2002; SIEMENS, 1998; STMICROELECTRONICS, 1999).

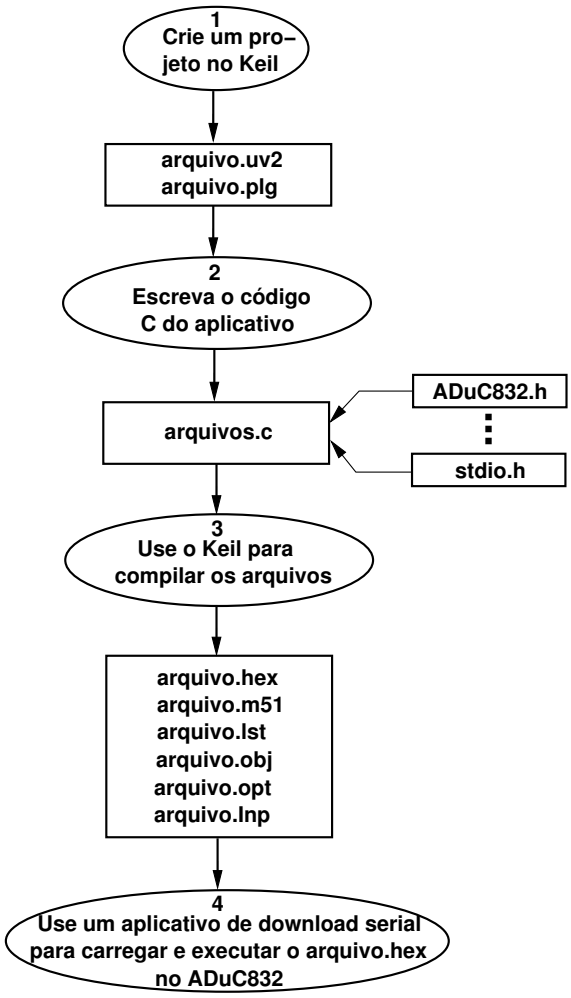

Figura 3.9: Seqüência de passos para construção de um aplicativo para o ADuC832.

3.7.1 Princípio de funcionamento

Na configuração padrão para um dispositivo, a SPI tem duas linhas de controle $\overline{SS}$ (*slave select*) e SCKL (*clock*) e duas linhas de dados MOSI (*Master Out Slave In*) e MISO (*Master In Slave Out*), como mostrado na Figura 3.10. A saída de dados MISO de um lado serve para leitura de retorno de dados do outro lado, além de possibilitar a ligação de diversos dispositivos em cascata (ESTL, 2002). Neste caso a saída de dados do dispositivo precedente forma a entrada de dados MOSI para o próximo dispositivo.

A comunicação SPI é do tipo mestre escravo (UBICOM, 2000; TEXAS INSTRUMENTS, 2003). O mestre fornece o sinal de *clock* (SCKL) e determina o estado da linha $\overline{SS}$ (NöLKER; KLEMENZ, 1999). Isto é, o mestre ativa o escravo e exige a comunicação com o mesmo. $\overline{SS}$ e SCKL são saídas do mestre e entradas para o escravo. A SPI especifica quatro sinais (MARTINEZ, 1997) básicos, descritos a seguir (UBICOM, 2000):

- **SCLK** - gerado pelo mestre, SCLK determina a velocidade da transferência de

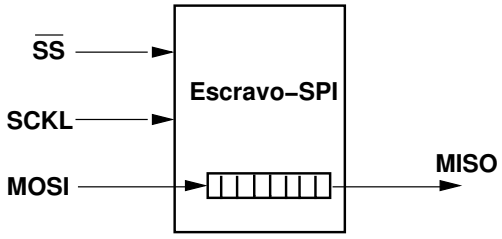

Figura 3.10: Escravo SPI

dados. Todos os dados são recebidos e transmitidos de forma síncrona com o *clock* interno do registrador de deslocamento e o *driver* de saída do sinal de *clock* (ESTL, 2002). SCLK é usado para sincronizar dados que estão sendo transferidos e recebidos pelas linhas MOSI e MISO. Cada *bit* é recebido ou transmitido em cada período de *clock*. Em ambos os modos mestre e escravo, dados são transmitidos sobre uma borda e amostrados sobre a outra (ANALOG DEVICES, 2002a);

- **MOSI** - leva dados do mestre para o escravo. A linha MOSI do mestre deve ser conectada na linha MOSI do escravo. Os dados são transferidos *byte* a *byte*, sendo o *bit* mais significativo de cada *byte* transferido primeiro (ANALOG DEVICES, 2002a; STMICROELECTRONICS, 1999; UBICOM, 2000). Os dados são transferidos na borda de subida do *clock* no registrador de deslocamento interno. Na borda de subida de $\overline{SS}$ a entrada de dados é armazenada nos registradores lógicos internos (ESTL, 2002);

- **MISO** - saída do registrador de deslocamento da porta serial da SPI (ESTL, 2002). A linha MISO do mestre deve ser conectada na linha MISO do escravo. A linha MISO leva dados do escravo para o mestre. Os dados são transferidos *byte* a *byte*, sendo o *bit* mais significativo de cada *byte* é transferido primeiro (ANALOG DEVICES, 2002a; STMICROELECTRONICS, 1999; UBICOM, 2000). Os dados são transferidos na borda de descida do *clock* no registrador de deslocamento interno;

- $\overline{SS}$ - ativa a interface SPI (ESTL, 2002). O escravo é selecionado quando o $\overline{SS}$ é declarado (nível lógico zero) pelo mestre. Tão logo, o $\overline{SS}$ esteja em nível alto, o escravo não aceita sinais de *clock* ou dados e a saída da linha MISO fica em estado de alta impedância (ESTL, 2002).

A interface SPI é baseada em um simples registrador de deslocamento[3] (NÖLKER; KLEMENZ, 1999) até um subsistema independente. O princípio básico do registrador de deslocamento está sempre presente. Código de comandos e dados são serialmente

[3]Normalmente o registrador de deslocamento é de oito bits ou múltiplo de oito (MOTOROLA, 2002; TEXAS INSTRUMENTS, 2003).

transferidos. Inseridos em um registrador de deslocamento e disponibilizado internamente para processamento paralelo (ESTL, 2002).

Na Figura 3.11 (SIEMENS, 1998) é ilustrado o princípio de funcionamento da comunicação SPI de um mestre e um escravo (MOTOROLA, 2002; UBICOM, 2000), idêntico ao usado na implementação TII.

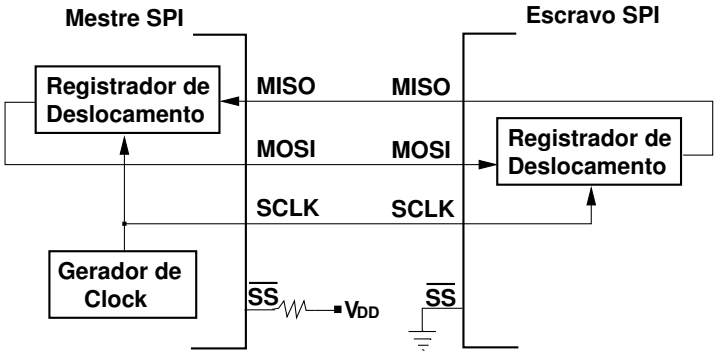

Figura 3.11: Diagrama de blocos da transferência mestre/escravo

3.7.2 Modos de operação da SPI

A interface SPI possui um par de parâmetros para configuração da polaridade e fase do sinal de *clock*. O parâmetro CPOL (*clock polarity*), determina a polaridade do sinal de *clock*. O parâmetro CPHA (*clock phase*) determina as bordas do sinal de *clock* sobre a qual, dados são amostrados e conduzidos (transmitidos/recebidos) (MOTOROLA, 2002). Cada um desses parâmetros possui dois estados, possibilitando quatro combinações distintas, como ilustrado na Figura 3.12 (TEXAS INSTRUMENTS, 2003).

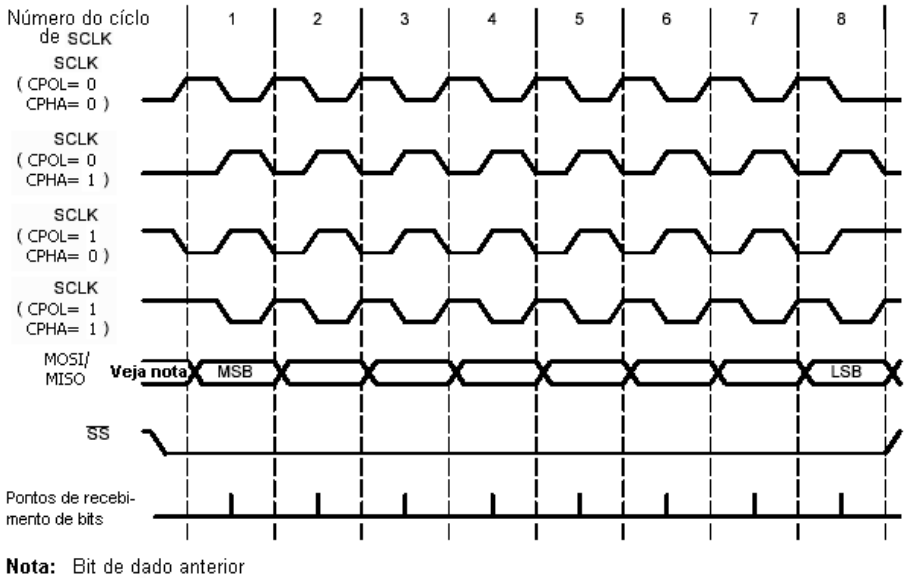

Figura 3.12: Sinais SPI para os quatro possíveis estados dos parâmetros CPOL e CPHA

A transferência dos *bits* ocorre na transição central de cada pulso de *clock*, como mostrado na última linha da Figura 3.12. Tanto o mestre como o escravo devem possuir a mesma configuração de CPOL e CPHA (ANALOG DEVICES, 2002a).

3.8 Conclusão

Neste capítulo foram apresentadas as principais características do *hardware* usado para implementação dos padrões IEEE 1451.1 (IEEE, 1999) (TINI modelo TBM390) (DALLAS, 2001) e IEEE 1451.2 (IEEE, 1997) (ADuC832) (ANALOG DEVICES, 2002a). Além disso foi apresentado, a nível de aspectos de funcionais, uma descrição da interface SPI (ESTL, 2002), usada como núcleo da comunicação da interface TII do IEEE 1451.2. Por fim, foi feita uma breve explanação sobre os conceitos da JNI (SUN, 2004), a fim de possibilitar o entendimento da biblioteca spi.tlib (MAXIM/DALLAS, 2001), desenvolvida em *software* para implementação da interface SPI do TINI.

Capítulo 4

Implementação dos padrões IEEE 1451.1 e IEEE 1451.2

4.1 Introdução

Neste capítulo é apresentado o ambiente de desenvolvimento, implementado para testes das implementações dos padrões IEEE 1451.1 (IEEE, 1999) e IEEE 1451.2 (IEEE, 1997) realizadas.

Aqui serão descritos os circuitos de condicionamento/medição e acionamento, bem como a especificação dos pinos da interface TII do lado do ADuC832 (ANALOG DEVICES, 2002a) e do TINI (DALLAS, 2001). Além disso, serão apresentados os resultados experimentais obtidos nos testes do STIM e NCAP.

4.2 Descrição global das implementações

O ambiente de desenvolvimento é constituído de: uma placa de desenvolvimento SAR *Eval Board* Rev A3 (ANALOG DEVICES, 2002b) com o microcontrolador ADuC832 (ANALOG DEVICES, 2002a), com a qual foi implementado o STIM (padrão IEEE 1451.2 (IEEE, 1997)); uma placa do TINI, modelo TBM390 (DALLAS, 2001), com a qual foi implementado o NCAP (padrão IEEE 1451.1 (IEEE, 1999)); um *proto-board* para montagem de circuitos de condicionamento de sinal; fonte de tensão DC de $5V$ e um PC, executando o SO *Windows* 98, onde são desenvolvidos os *software* para o ADuC832 e o TINI. Além dos circuitos de medição, para o sensor de temperatura LM35 (NATIONAL SEMICONDUCTOR, 2000) e circuitos de acionamento para acionar um pequeno ventilador.

O microcontrolador ADuC832 é programado em linguagem C e os aplicativos do TINI em linguagem Java ou em Java e Assembler, possibilitando a utilização dos conceitos da teoria de JNI (*Java Native Interface*) (SUN, 2004) para implementação de métodos nativos

(MN) (DALLAS, 2002b, 2002c). A teoria de JNI foi usada para entender o funcionamento de uma biblioteca nativa desenvolvida em *assembler* para implementar a interface SPI do TINI por *software*. Além disso, o TINI proporciona *interface* para a rede *Ethernet*, já bastante usada como rede de controle e medições distribuídas (BURCH; EIDSON; HAMILTON, 2000; LEE; SCHNEEMAN, 2000; MANDERS; BARFORD, 2000; MOORE; ROSS; JOHNSON, 1999; SCHNEEMAN, 1999; SVEDA; VRBA, 2002a, 2002b).

Os detalhes da especificação dos pinos para implementação do NCAP e do STIM com os detalhes da montagem do ambiente de desenvolvimento serão doravante descritos.

4.3 Acesso aos pinos e portas do microcontrolador

Nessa seção será apresentado a especificação dos pinos do ADuC832 e do TINI, para implementação TII do STIM e do NCAP, respectivamente.

4.3.1 Pinos e portas do TINI

Todos os pinos de portas do TINI pertence a uma das 6 $(0, 1, ..., 5)$ portas do microcontrolador (DS80C390 (MAXIM/DALLAS, 2003c)) do TINI. Uma porta é um grupo de oito pinos. Assim, por exemplo, a porta 3 designada por $P3$ é formada por um grupo de 8 pinos $[p3.0 - p3.7]$. As portas 0, 1, 2 e 4 são destinadas ao barramento de dados, barramento de endereço e sinais de *chip enable*. Isto faz os pinos das portas 3 e 5 candidatos para uso como pinos de entrada e saída de propósito geral (DALLAS, 2001).

É importante conhecer bem, como os pinos das portas são avaliados para uso com seus aplicativos, pois muitos pinos de portas servem para dois e algumas vezes três, propósitos distintos. Isso requer um estudo cuidadoso dos recursos intrínsecos dos I/Os usados, tais como, serial e CAN. Por exemplo os 4 pinos de alta ordem da porta 5 ($p5.4 - 5.7$), seu uso normal destinado para decodificar *hardware* interfaceado pelo barramento do microcontrolador. Se o sistema não usa qualquer dos sinais PCE (*Peripheral Chip Enable*) para propósito de decodificação lógica, então todos esses 4 pinos podem ser usados como pinos de propósito geral[1]. Os pinos PCE foram usados na implementação da interface SPI.

4.3.2 A classe BitPort do TINI

O acesso individual dos pinos das porta é alcançado usando a classe `BitPort`, contida no pacote `com.dalsemi.system`. O seguinte construtor é usado para anexar um objeto

[1]Com todos ou nenhum sinal PCE. Se o sistema usa qualquer um dos sinais PCE como *chip enable* verdadeiro, nenhum dos sinais remanescentes podem ser usados como pinos de porta de propósito geral.

`BitPort` ao pino da porta específica.

```
public BitPort(byte bitname)
```

O pino da porta é especificado pelo parâmetro `bitname`. Valores válidos para `bitname` são definidos como constantes públicas na classe `BitPort`. Os nomes são constantemente formados concatenando o *string* `Port`, seguido pelo número da porta para o *string* `Bit`, seguido pela posição do *bit* (ou pino) da porta. Assim, por exemplo, a seguinte instrução cria um novo objeto `BitPort` anexado ao pino 3 da porta 5 do microcontrolador.

```
BitPort bp = new BitPort(BitPort.Port5Bit3);
```

Os métodos `set` e `clear` são usados para controlar o estado do pino:

```
public void set()
public void clear()
```

Quando o método `clear` é invocado, o pino da porta é levado para nível lógico baixo (0). Por outro lado, quando um método `set` é invocado o pino é levado para nível lógico alto (1). A invocação do método `read` retorna o valor amostrado pelo pino da porta. Se durante o tempo de amostragem o pino está alto (lógica 1), `read` retorna 1. Caso contrário, `read` retorna 0. O método `set` pode ser invocado antes da primeira chamada do `read`. Também, se em qualquer tempo o pino da porta é levado para baixo via invocação do método `clear`, o método `set` deve ser invocado para permitir circuitos externos ativem o pino antes de invocar `read`.

```
public int read()
```

4.3.3 Linhas físicas TII do STIM

A verificação dos pinos e portas do microcontrolador para cada linha TII do lado do STIM/ADuC832 e o papel desempenhado por cada pino destas portas na implementação do protocolo de comunicação é mostrado na Tabela 4.1.

Tabela 4.1: Especificação de pinos do ADuC832 para implementação da interface TII do STIM, padrão IEEE 1451.2

Linha TII	**Pinos do ADuC832**	**Placa do ADuC832**	**Descrição**
DOUT	MISO	J3, pino 3	*Master In, Slave Out* - pino de I/O de dados. Parte da porta da interface SPI.
DIN	MOSI	J3, pino 4	*Master Out, Slave In* - pino de I/O de dados. Parte da porta da interface SPI.
DCLK	SCLOCK	J3, pino 5	Clock serial I/O. Parte da SPI.
NIOE	*Slave Select*	J3, pino 2	Pino de entrada *Slave Select*. Parte da interface SPI. Quando usado indica que a SPI está no modo escravo, isto é o clock serial será controlado externamente.
NTRIG	Porta 3.2 (INTO)	J7, pino 3	Pino de porta de I/O de propósito geral, configurado como uma entrada. A razão para usar NTRIG neste pino particular é que ele pode ser configurado (via software) como uma linha de interrupção externa.
POWER	Vcc	*Power Rail*	Fonte de potência para o STIM - por definição a potência para o STIM deve fornecida pelo NCAP.
COMMOM	Gnd	*Ground Rail*	Terra de referência para o STIM.
NACK	Porta 3.6	J7, pino 5	Pino de I/O de propósito geral, configurado como uma saída.
NSDET	Não conectado	Conectada via resistor de 10 $k\Omega$ para a terra	A linha NSDET é ativa baixa, e deve de ser conectado ao terra do lado do STIM.
NINT	Porta 3.5	J7, pino 6	Pino de I/O de propósito geral, configurado como uma saída.

4.3.4 Linhas físicas TII do lado do NCAP/TINI

A verificação dos pinos e portas do microcontrolador para cada linha TII do lado do NCAP/TINI e o papel desempenhado por estas portas na implementação do protocolo de comunicação é ilustrado na Tabela 4.2.

Tabela 4.2: Especificações de pinos do TINI para implementação da interface TII do NCAP, padrão IEEE 1451.1

TII Linha	**Pinos μC $N^{\underline{o}a}$/Nome**	**Pinos-TBM390 $N^{\underline{o}a}$/Nome**	**Descrição**
DOUT	MISO 15/P5.6	28/$\overline{PCE2}$	*Master In, Slave Out* pino de I/O de dados. Parte da interface SPI.
DIN	MOSI 16/P5.5	29/$\overline{PCE1}$	*Master Out, Slave In* pino de I/O de dados, parte da interface SPI.
DCLK	17/P5.4	30/$\overline{PCE0}$	Clock serial I/O. Parte da SPI.
NIOE	14/P5.7	27/$\overline{PCE3}$	Pino de saída, *Slave Select*. Parte da SPI. Habilita transferênça de dados.
NTRIG	20/P5.1	11/CRX	I/O bi-direcional de propósito geral. Usado para realização de gatilhamento.
POWER	10/P3.4	18/$\overline{DRST}$	I/O bi-direcional de propósito geral. Usado para alimentar os circuitos de controle da interface do STIM.
COMMOM	9,25,41,57/Gnd	3/4/5	Terra para circuitos digital.
NACK	5/P3.1	21/TX	I/O bi-direcional de propósito geral. Usado pelo NCAP para reconhecer transporte de dados.
NSDET	21/P5.0	10/CTX	I/O bi-direcional de propósito geral. Usado pelo NCAP para detectar evento de inserção.
NINT	P3.3/7	23/$\overline{EXTINT}$	Interrupção externa de propósito geral.

4.3.5 Ambiente de desenvolvimento

O ambiente de desenvolvimento é constituído por uma placa de desenvolvimento SAR *Eval Board* Rev A3 (ANALOG DEVICES, 2002b) com o microcontrolador ADuC832 que é usada para implementação do padrão IEEE 1451.2. Os pinos do ADuC832 usados para implementação da *interface* TII são descritos na Tabela 4.1 e são mostrados juntamente com os pinos para ligação do sensor e do atuador na Figura 4.1; uma placa TINI (DALLAS, 2001), modelo TBM390, onde os aplicativos desenvolvidos podem disponibilizar, em uma página *web*, os dados adquiridos resultantes de medições realizadas por sensores, conecta-

dos aos ADs do ADuC832 e o estado e/ou limites de referência de atuadores conectados aos DAs do ADuC832. O TINI foi usado para implementar o NCAP. A especificação dos pinos para implementação da *interface* TII do lado do NCAP com o TINI são mostrados na Tabela 4.2; um *proto-board* para montagem de circuitos de medição, acionamento e condicionamento de sinais; 1 fonte de tensão DC de $5V$, 1 multímetro, 1 osciloscópio e um PC com sistema operacional *windows*, usado no desenvolvimento dos aplicativos para o TINI e ADuC832. Os detalhes de montagem do ambiente de desenvolvimento são mostrados na Figura 4.1.

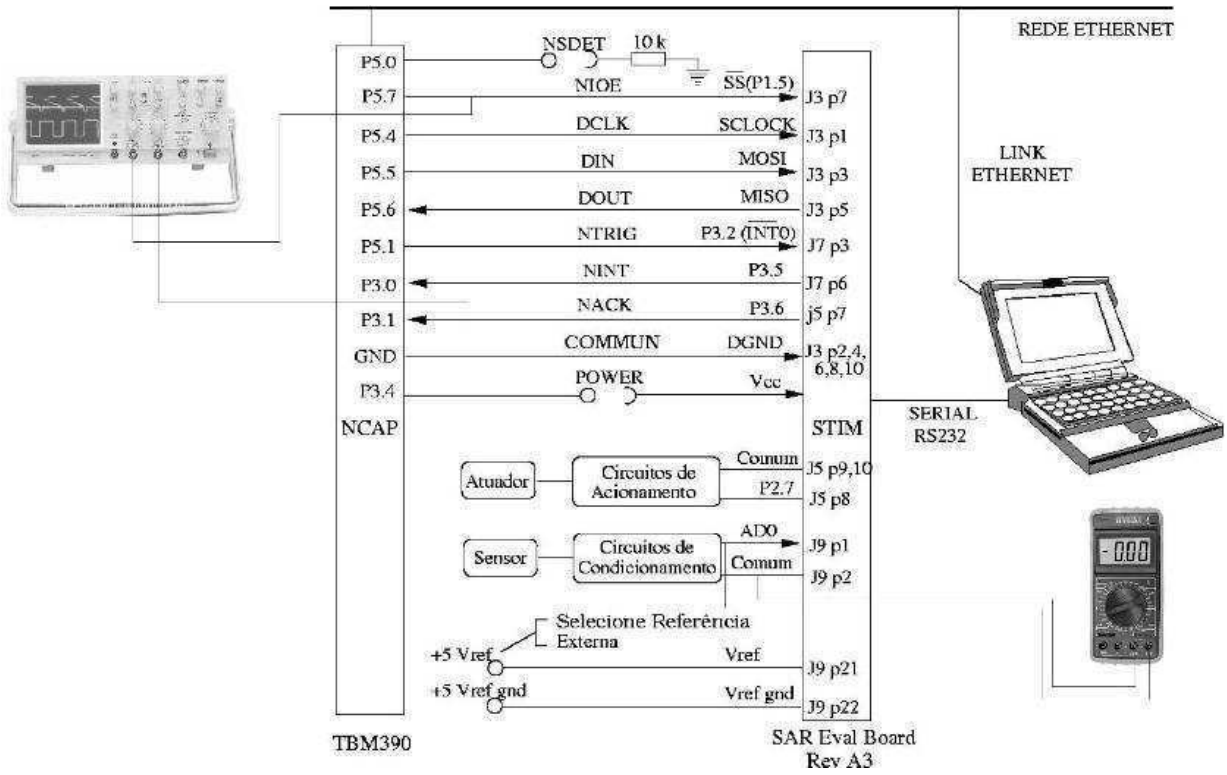

Figura 4.1: Conexões físicas feitas na placa *ADuC8xx SAR Evaluation Board Reference Guide* e no TINI

Observe que na parte superior da Figura 4.1 é mostrada em detalhes a especificação da interface TII, tanto do lado do STIM como do lado do NCAP. Na parte central dessa Figura é mostrado o diagrama de blocos dos circuitos de acionamento de um transdutor. Na parte inferior da mesma também é mostrado o diagrama de blocos dos circuitos de condicionamento de sinais. Os detalhes de implementação desses circuitos podem ser vistos respectivamente, nas Figuras 4.2a e 4.2b.

A conexão serial RS232 que conecta o PC ao STIM serve para carregar os aplicativos desenvolvidos para implementação do STIM no ADuC832, para visualizar o conteúdo de variáveis via *HyperTerminal* do *Windows*, além de possibilitar a depuração dos aplicativos. O *link Ethernet* é para conectar o PC e o TINI à rede *Ethernet*.

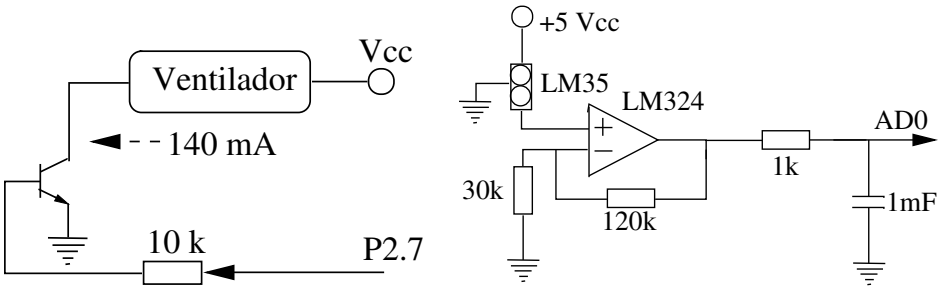

Figura 4.2: (a) Circuito de acionamento de um pequeno ventilador. (b) Circuito de condicionamento de sinal para um sensor LM35.

4.3.6 Interface TII do lado do NCAP

A associação dos pinos do TINI ao *software* é feita instanciando a classe `BitPort` da API. Por exemplo, para o pino da linha NTRIG, tem-se:

```
BitPort NTRIG = new BitPort(BitPort.Port5Bit1);
```

Em seguida são definidos métodos para declarar, negar e ler os pinos das linhas da interface TII. Por exemplo, para declarar o pino da linha NTRIG tem-se o seguinte método:

```
public void assertNTRIG(){NTRIG.clear();}
```

e para negar tem-se:

```
public void negateNTRIG(){NTRIG.set();}
```

Um exemplo de método para leitura do pino da linha NACK é:

```
public int readNACK(){return NACK.read();}
```

Os pinos das linhas DCLK, DIN e DOUT são controlados pela interface SPI. O pino da linha NINT no TINI é definido como uma interrupção externa, a qual ocorre quando o STIM declara NINT. Um exemplo de método para a leitura de quadro do Meta-TEDS, pelo NCAP, é mostrado a seguir. Esse método é executado após um evento de inserção.

```
public void readMetaTEDS(){
  int numBytes = 0;
  for(int count=0; count<4; count++){
     //Verifica estado da linha NACK e espera NACK mudar seu estado
     tii.toggleNACK(count);
     //Leia byte e armazene no array metaTEDS[]
     this.metaTEDS[count] = readByte();
```

```
    } //Fim do bloco for
    //Rotina que calcula o numero de bytes do TEDS
    numBytes = 16777216*metaTEDS[0] + 65536*metaTEDS[1] +
    + 256*metaTEDS[2] + metaTEDS[3];
    numBytes = numBytes + 4;
    for(int count=4; count < numBytes; count++){
       tii.toggleNACK(count);
       this.metaTEDS[count] = readByte();
    } //Fim do bloco for
}
```

4.4 Testes funcionais

Ao término da implementação do STIM e do NCAP, foi iniciada a fase de testes dos *softwares* desenvolvidos diretamente no ADuC832 e no TINI.

4.4.1 Software do STIM

Inicialmente foi testada a leitura dos TEDS da implementação do STIM isoladamente no ADuC832, a partir do *Hyper Terminal* do *windows*. Este *software* é composto de 4 arquivos escritos em linguagem C e 5 arquivos bibliotecas, também escritos em linguagem C, onde são definidos os tipos de dados e os protótipos das funções usadas. Cada um dos quatro arquivos implementa partes do padrão IEEE 1451.2, a saber: gatilhamento, interface TII, blocos de endereçamentos e funções e os TEDS (ANALOG DEVICES, 1999).

O fluxograma do *software* desenvolvido na implementação do STIM é mostrado na Figura 4.3.

4.4.2 Software do NCAP

O *software* desenvolvido para implementação do NCAP foi feito em Java e é composto de 17 classes, sendo que destas 4 implementam o núcleo do NCAP. Estas 4 classes são responsáveis pela implementação da interface TII; dos métodos de leitura e escrita e do bloco transdutor, onde é armazenada toda a informação dos transdutores, obtidas via leitura dos TEDS. As outras 13 classes são responsáveis pela implementação da interface gráfica utilizada e foram desenvolvidas pelo fabricante para uma aplicação com rede *1-wire* (MAXIM/DALLAS, 2002d) e reutilizadas neste trabalho. O fluxograma do *software* desenvolvido é ilustrado na Figura 4.4.

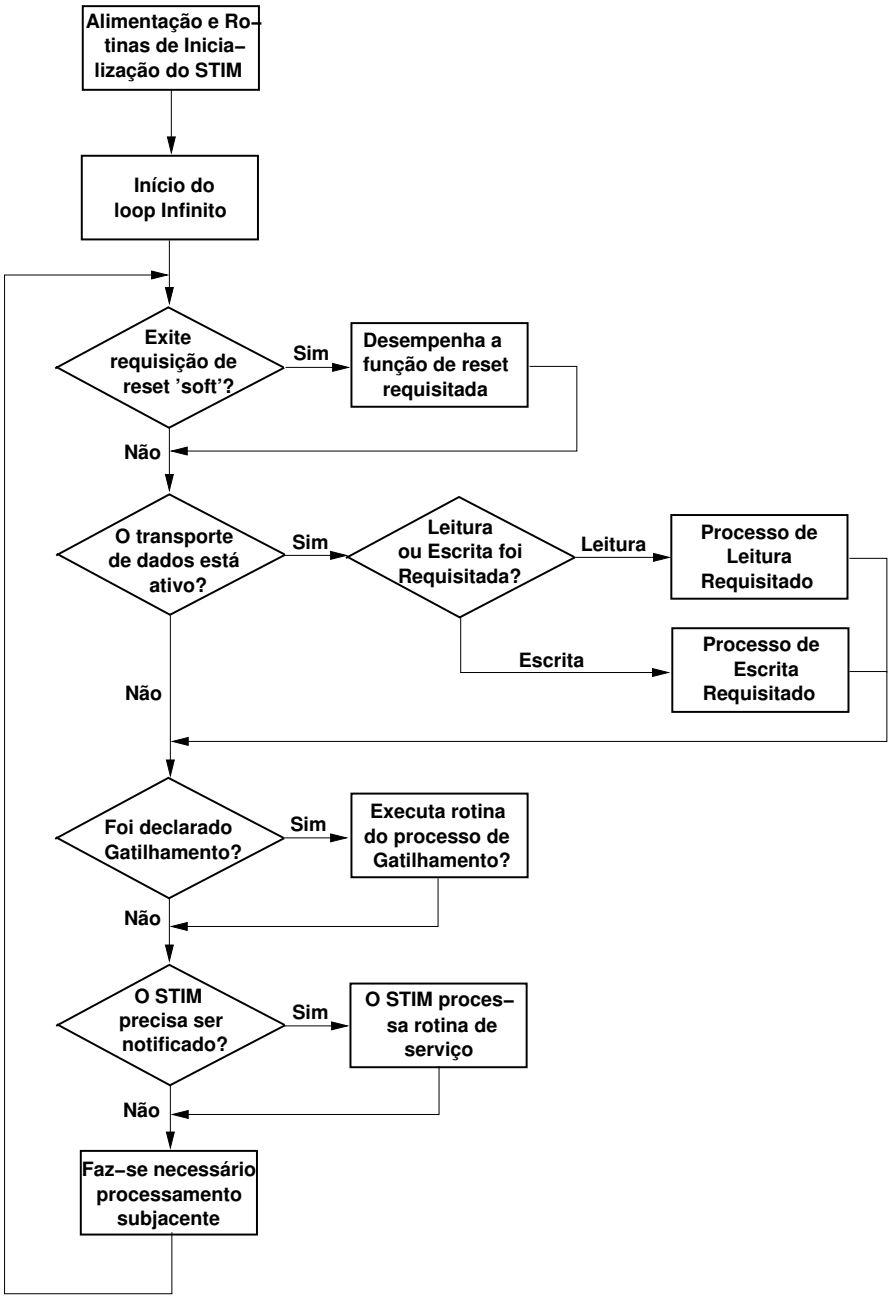

Figura 4.3: Fluxograma do *software* desenvolvido na implementação do STIM.

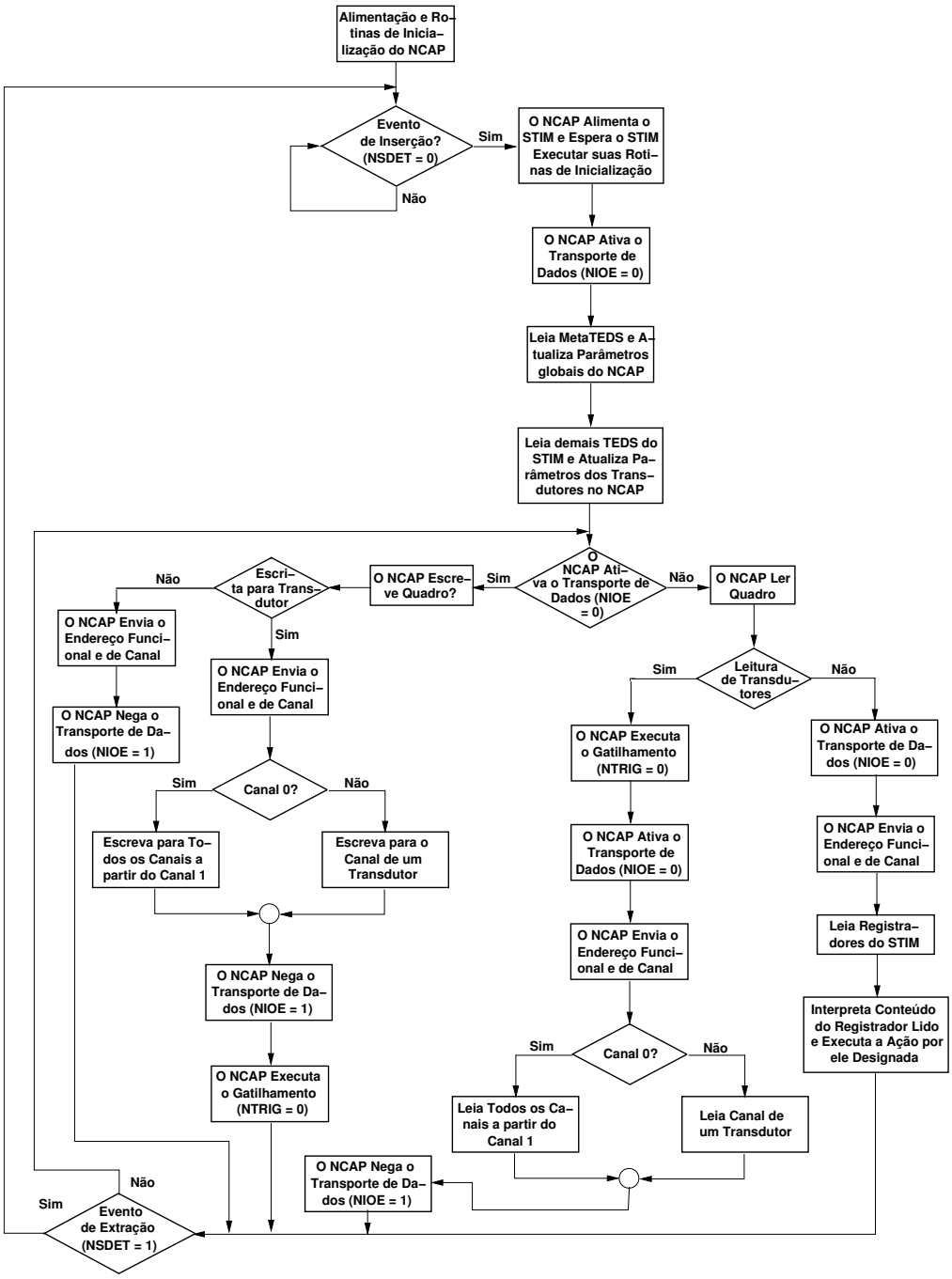

Figura 4.4: Fluxograma do *software* desenvolvido na implementação do NCAP.

4.5 Resultados experimentais

Os sinais da *interface* TII, obtidos com um osciloscópio através da monitoração dos pinos das linhas da mesma, são mostrados na Figura 4.5. Estes sinais foram obtidos dois a dois e montados em uma mesma figura, para possibilitar a comparação com a especificação do padrão IEEE 1451.2 que define a *interface* TII e estão de acordo com a especificação

do padrão IEEE 1451.2, verificando assim a integridade das implementações realizadas. Esta Figura 4.5 pode ser comparada com as Figuras 2.8 e 2.12 do padrão apresentado no capítulo 2 deste documento.

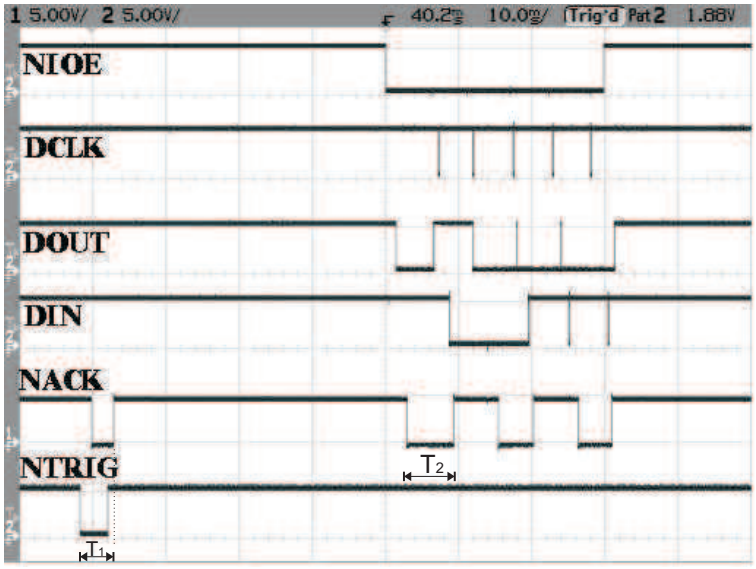

Figura 4.5: Diagrama de temporização das linha da interface TII.

4.5.1 Indicadores de desempenho

Através de medições realizadas com o osciloscópio foram verificados os seguintes tempos de leitura: 450,0 ms (milissegundos) para o Meta-TEDS de 76 *bytes*; 570,0 ms para cada TEDS de canal de 96 *bytes*; 27,0 ms para canal 0 de 5 *bytes* e 21,0 ms para canal 1 de 4 *bytes*. O tempo médio de leitura de um *byte* é de 5,9 ms. O tempo médio para a realização de um gatilhamento, verificado para os canais implementados foi de 4,0 ms. Na Figura 4.5, T_1 e T_2 são respectivamente, o tempo de gatilhamento e o tempo de leitura de um *byte* pelo NCAP.

O alto tempo, gasto para leitura dos TEDS, gatilhamento, leitura de um *byte* obtidos, deve-se à baixa taxa de transferência, cerca de no máximo 100 Kbit/s, obtida com a implementação da SPI em assembler, comparada à taxa de comunicação da SPI, implementada a nível de camada física, que é da ordem de alguns megabits por segundos. Outro fator que contribui para a redução na velocidade de execução do código do NCAP, executado no TINI, é a baixa taxa de execução de *byte codes* da JVM do TINI. Este pode ser visto comparando o tempo de escalonamento de *threads*, pela JVM do TINI, que é de 2 milissegundos, comparada com outros dispositivos também programados em Java, como JStik (SYSTRONIX, 2004), que é de 1 microssegundo (SYSTRONIX, 2005).

Na implementação do STIM foram utilizados: 178 *bytes* de memória RAM de 256 disponíveis; 268 *bytes* de memória *flash/EE* de dados de 4096 disponíveis, usados para

armazenar os TEDS e 6997 *bytes* de memória *flash/EE* de programa de um total de 68993 *bytes*.

Os arquivos a serem carregados, via FTP, no TINI são: TempApp.jar com 19698 *bytes*, TempApp.tini com 19233 *bytes* e index.html de 185 *bytes* de um total de 512 *kBytes* memória RAM. Apenas o arquivo TempApp.tini é diretamente executado.

4.6 Interface gráfica do NCAP

Nesta seção será descrito o *software* do NCAP do ponto de vista da comunicação entre servidores *web* e clientes *web*. Além disso são mostradas as relações entre os blocos de *software* desenvolvidos e a interface gráfica onde os dados dos transdutores inteligentes são mostrados.

4.6.1 Relação entre *applets* e *servlets*

A estação de monitoração ligada à rede *Ethernet* com *browser* dotado de uma JVM que suporte *applets* Java é parte indispensável do sistema de monitoração dos transdutores inteligentes. *Applets* são pequenos aplicativos embarcados em páginas *web*, o qual pode ser usado para construir elementos GUI (*Graphical User Interfaces*) complexos sobre uma página *web* com transferência síncrona de informação de servidores *web* para clientes *web*. Tal mecanismo permite que dados de transdutores sobre a página *web* sejam atualizados automaticamente. *Applets* também podem ser usados em conjunto com aplicativos *servlets* para transmitir informações de clientes *web* para um servidor *web*. *Servlets* são executados sobre um servidor *web* e são criados para listar *strings* de comandos enviados pelos *applets*. O processo de comunicação entre *applets* e *servlets* é ilustrado na Figura 4.6 (COSTA, 2002).

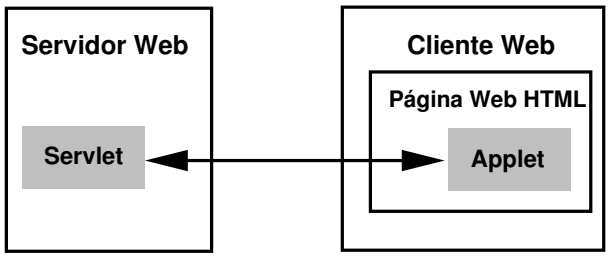

Figura 4.6: Processo de comunicação entre *applets* e *servlets*.

A relação complementar entre *applets* e *servlets* forma a base da rede de comunicação entre a estação de monitoração e o NCAP. Inicialmente o NCAP é ativado como um servidor *web* HTTP (*HiperText Transfer Protocol*) servindo o *applet* embarcado na página *web* da estação de monitoração. Depois um aplicativo *servlet* é executado no NCAP

fornecendo um canal de comunicação bidirecional entre a estação de monitoração (cliente *web*) e o NCAP (servidor *web*) (COSTA, 2002).

4.6.2 Estrutura do Software

A estrutura dos módulos de *software* desenvolvidos para implementar a modularidade e interoperabilidade do padrão IEEE 1451 pode ser vista na Figura 4.7. De um lado está a estação de monitoração executando a GUI Java que atualiza os dados recebidos da comunicação entre o *applet* executado no PC e o *servlet* executado no TINI. Do outro lado, entre o TINIm300 e o ADuC832, ocorre a execução dos protocolos TII, responsáveis pela conexão entre o NCAP e o STIM.

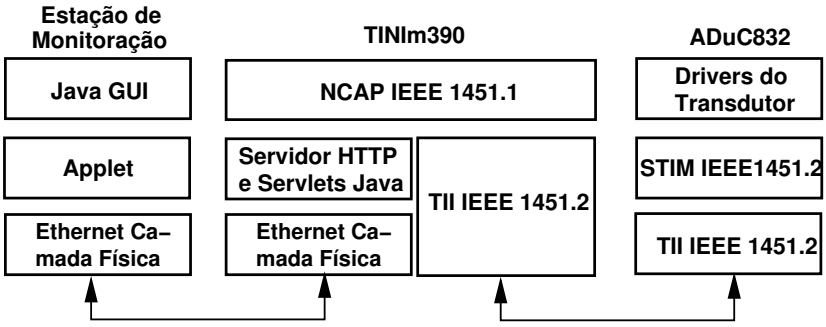

Figura 4.7: Estrutura dos módulos de *software* desenvolvidos para a implementação do NCAP.

4.6.3 Aplicativo básico

A interface gráfica disponibiliza a temperatura do sensor LM35 usado na implementação do STIM, na forma gráfica desenhada por um *applet* executado diretamente de uma página *web*, tão logo a aquisição dos dados seja feita. Inicialmente a tensão é amostrada na saída do circuito de condicionamento de sinais pelo conversor AD do ADuC832 e após gatilhamento é enviada ao NCAP via interface TII. No NCAP os *bytes* recebidos são convertidos em temperatura usando-se a relação de equivalência da especificação do sensor de temperatura e mostrada na forma gráfica e em um *display*. Nesta interface o valor da temperatura pode ser visto na escala *Celsius* ou *Fahrenheit*. Além disso, é possível ajustar a escala de tempo e temperatura nos respectivos eixos do tempo (horizontal) e da temperatura (vertical). A interface também apresenta um botão para controle da taxa de amostragem dos dados na tela, a qual pode chegar a uma amostra por segundo. Esta interface foi desenvolvida em Java pelo fabricante do TINI (MAXIM/DALLAS, 2002d) e alterada para os respectivos testes. O resultado obtido é mostrado na Figura 4.8.

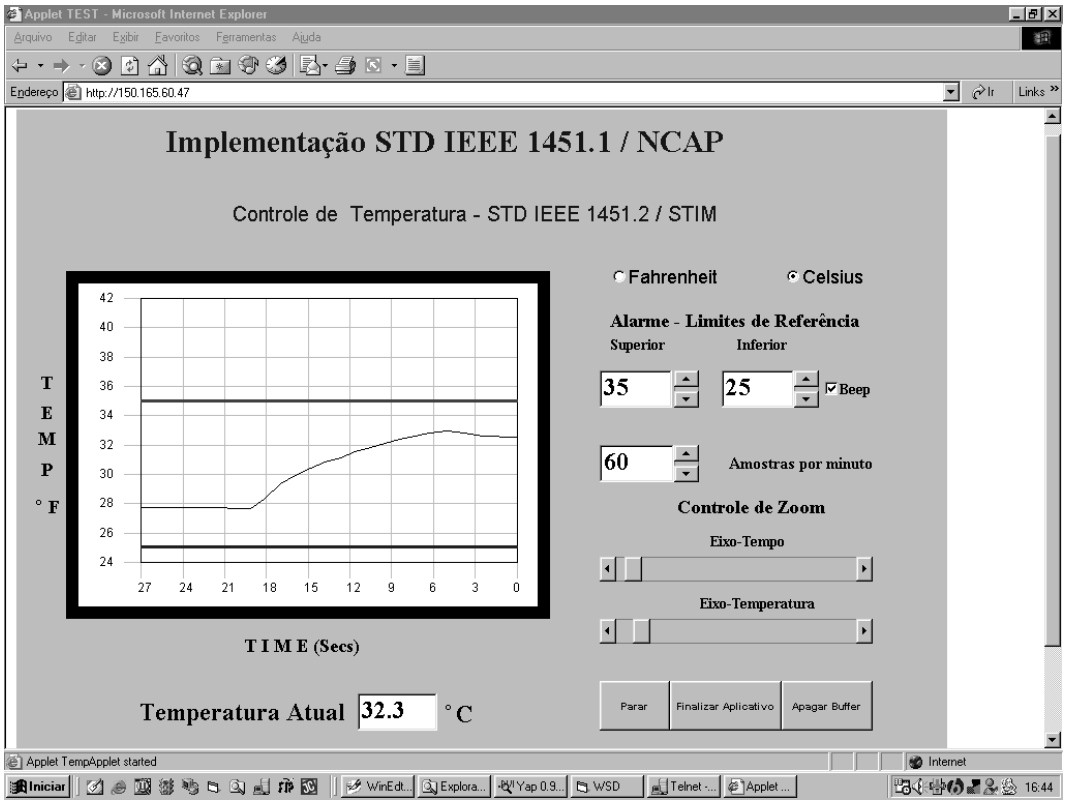

Figura 4.8: Página *web* mostrando interface gráfica da implementação IEEE 1451.1

O acesso à página *web* executada diretamente no TINI é obtida digitando-se o IP (*Internet Protocol*) na barra de endereço do *browser* de um PC conectado à *intranet/internet*. Esta interface pode ser vista de qualquer ponto de *internet* do mundo. O sinal gráfico entre as linhas azul e vermelha representa a temperatura e deve ser praticamente reta horizontal. Nesta Figura ela está curva pelo fato de uma variação da temperatura ter sido provocada para ilustrar o funcionamento do sistema. Isto é, o sensor foi aquecido, levando a uma variação significativa da temperatura. Instantes depois a fonte de aquecimento foi retirada e a temperatura medida começou a reduzir, tendendo à temperatura ambiente.

4.7 Conclusão

Neste capítulo foi apresentada o ambiente de desenvolvimento, usado para desenvolvimento e testes da implementação dos padrões IEEE 1451.1 (IEEE, 1999) e IEEE 1451.2 (IEEE, 1997). Também foi apresentada a especificação dos pinos da interface TII do lado do ADuC832 (ANALOG DEVICES, 2002a) e do TINI (DALLAS, 2001). Bem como os os circuitos de condicionamento, medição e acionamento usados, culminando com os resultados experimentais obtidos.

Estes resultados experimentais, obtidos em laboratório estão de acordo com a especificação. Isso abre os horizontes para implementação de diversos tipos de transdutores

inteligentes segundo esses padrões, uma vez que as funções do STIM e estão testadas e funcionando, o desenvolvimento de novos transdutores inteligentes fica restrita ao desenvolvimento dos TEDS, os quais são responsáveis pela inteligência do transdutor. Outro aspecto a ser considerado é a redução no custo de desenvolvimento das interfaces do NCAP, visto que o esforço de desenvolvimento ficará restrito à implementação das interfaces, em sua grande maioria reutilizadas de outras aplicações.

Um exemplo de re-utilização de interface é mostrado nesse trabalho ao ser usado uma interface desenvolvida pelo fabricante do TINI. Dessa maneira o esforço de desenvolvimento ficará restrito praticamente ao desenvolvimento e melhoria das interfaces. Frente a esse cenário a redução no custo de aquisição de tais transdutores é um fato, uma vez que os dispositivos usados para implementá-los não passa da cifra dos 100 a 200 dólares.

Capítulo 5

Conclusões

Este trabalho apresentou as etapas de construção de um ambiente para desenvolvimento de aplicações baseadas em transdutores inteligentes segundo a especificação dos padrões IEEE 1451.1 e IEEE 1451.2. Para implementação foram utilizadas duas placas: a SAR *Eval Board Rev A3* com o microcontrolador ADuC832 para implementação do STIM (padrão IEEE 1451.2) para aquisição de dados do sensor e acionamento de atuador e a TBM390 do TINI para proporcionar interface com a rede *Ethernet*, foi utilizada para implementação do NCAP (padrão IEEE 1451.1). Para o STIM desenvolvido foram implementados apenas os TEDS mandatórios (Meta-TEDS e TEDS de canais).

Dentro do escopo proposto nesse trabalho, os resultados obtidos foram satisfatórios não apenas pelo fato da construção do ambiente de desenvolvimento para implementação de transdutores inteligentes, mas por possibilitarem avanços significativos no trabalho com sistemas embarcados, voltados para instrumentação eletrônica no tocante a supervisão, medições e controle distribuídos via rede *Ethernet*. Embora seja sabido que a *Ethernet* não tenha sido projetada para comunicações em tempo real, uma vez que seu mecanismo de acesso ao meio não garante a transmissão de uma mensagem com restrição de tempo. Para aplicações com constante de tempo longa, a *Ethernet* satisfaz as restrições de tempo e portanto pode ser usada nestas aplicações, possibilitando a construção de sistemas distribuídos para supervisão, medições e controle.

A principais contribuições deste trabalho foram as implementações do NCAP (padrão IEEE 1451.1) com o TINI e do STIM (padrão IEEE 1451.2) com o ADuC832, possibilitando o desenvolvimento de aplicações distribuídas para controle e supervisão a um custo baixíssimo. A principal vantagem na utilização do TINI é que ele é programado em Java. Isto possibilita o desenvolvimento de aplicações incrementadas com o ferramental disponível para Java. Entretanto, o fato do TINI não ter interface SPI a nível de camada física foi um fator bastante complicador durante a implementação. Outra desvantagem, neste caso, é a baixa taxa de transferência, cerca de no máximo 100 Kbit/s, obtida com a

implementação da SPI em assembler, comparada com a SPI a nível de camada física que é da ordem de alguns megabits por segundos.

Para trabalhos futuros pretende-se:

- desenvolver os TEDS não mandatórios do padrão IEEE 1451.2 para os transdutores inteligentes já desenvolvidos, para possibilitar, por exemplo, que as informações do fabricante do transdutor sejam identificadas e mostradas na interface homem-máquina do NCAP;
- aprimorar as interfaces gráficas da implementação do padrão IEEE 1451.1, para possibilitar a visualização de uma maior quantidade de informações fornecidas pelos TEDS;
- montar a rede de transdutores inteligentes segundo os padrões IEEE 1451.1 e 1451.2, para possibilitar a implementação de aplicações distribuídas usando como suporte de rede a rede *Ethernet*;
- implementar o NCAP com algum *hardware* que tenha a SPI a nível de camada física. A sugestão é que se utilize o JStik (SYSTRONIX, 2004), o qual é programado em Java e têm SPI a nível de camada física, com taxa transferência de 6 Mbits;

Apêndice A

Redes de transdutores

A.1 Introdução

As redes de comunicação passaram a fazer parte do cenário dos sistemas de controle digital nos anos setenta visando atender à demanda da indústria automotiva. O uso de redes era eminentemente motivado pelas seguintes razões: redução do custo de cabeamento, modularização dos sistemas e flexibilidade na configuração do sistema (JOHNSON, 1997). Desde então, diversos tipos de redes e protocolos foram desenvolvidos para suprir uma demanda crescente e diversificada de aplicações. Exemplos de sistemas de controle que utilizam redes são encontrados na automação industrial, em sistemas de veículos inteligentes e em aeronaves (CHOW; TIPSUWAN, 2001).

Os SCRs (Sistemas de Controle em Redes) representam uma evolução dos sistemas de controle digitais que utilizam redes de comunicação. Esta evolução se deu basicamente no ramo da eletrônica, o que possibilitou integrar interfaces de rede a sensores e atuadores, além de possibilitar que sensores e atuadores realizassem determinados tipos de processamento, como auto-calibração, identificação *plug-and-play* quando conectados à rede, tratamento de entradas espúrias (ruído). Desde então, tais dispositivos puderam ser ligados diretamente em rede.

As redes utilizadas para fins de controle são caracterizadas pelo transporte freqüente de uma grande quantidade de pequenos pacotes de dados entre um grande número de nós. Além disso, estas redes devem atender a requisitos temporais críticos.

Dentre os vários protocolos existentes que podem ser usados em SCR muitos são padrões proprietários. Estes protocolos são classificados em duas categorias principais: barramentos de campo e redes de propósito geral.

- Barramentos de campos (*Fieldbuses*) - FIP (*Factory Instrumentation Protocol*), PROFIBUS (*Process Fieldbuses*) (ASSOCIAçãO PROFIBUS BRASIL, 2000), *Foundation Fieldbus*, CAN (*Controller Area Network*) (GMBH, 1991; LAWRENZ, 1997) e

DeviceNet;

- Redes de propósito geral - IEEE 802.3 (*Ethernet*), IEEE 802.11 e *Bluetooth* (HAARTSEN, 2000).

Dentre estas tecnologias as utilizadas para redes sem fio, IEEE 802.11 e *Bluetooth*, vêm despertando um interesse crescente, evidentemente justificado pelo fato dos custos de cabeamento representarem uma grande parte do custo de um sistema de controle. Além disto, enquanto o custo dos sensores e atuadores têm diminuído ao longo dos anos, o custo do cabeamento tem permanecido praticamente constante. Um outro atrativo destas tecnologias é a possibilidade delas poderem ser usadas em redes *ad hoc* (ROYER; TOH, 1999). As redes *ad hoc* são caracterizadas por não possuírem uma configuração pré-estabelecida, de modo que sua topologia não é fixa. Essa característica, juntamente com a mobilidade dos dispositivos do sistema, permite a instalação de sistemas de controle em ambientes hostis à presença humana e sujeitos a mudanças de configuração freqüentes. Entretanto, estas tecnologias de comunicação sem fio apresentam alguns problemas que afetam o seu desempenho se comparada às tecnologias de redes com fios, tais como: propagação dos sinais através de múltiplos percursos, desvanecimento, forte queda na potência do sinal quando esse tem obstáculos a ultrapassar e elevada taxa de erro de *bit* (NASCIMENTO, 2004).

A evolução dos transdutores e das redes ocorreu, em parte, graças à evolução da tecnologia da informação que tem sido determinante no desenvolvimento da tecnologia da automação, a qual alterou hierarquias e estruturas no ambiente dos escritórios e chegou nos mais diversos setores do ambiente industrial, contemplando desde indústrias de processo e manufatura até prédios e sistemas logísticos. A capacidade de comunicação entre dispositivos e o uso de mecanismos padronizados, abertos e transparentes são componentes indispensáveis ao conceito de automação atual. A comunicação vem se expandindo rapidamente, integrando todos os níveis hierárquicos. De acordo com as características da aplicação e do custo máximo a ser atingido, uma combinação gradual de diferentes sistemas de comunicação, tais como Ethernet, PROFIBUS e AS-Interface, oferece as condições ideais de redes abertas em processos industriais (ASSOCIAçãO PROFIBUS BRASIL, 2000).

A seguir é apresentada uma descrição sucinta de algumas redes e barramentos de campo usados para controle e supervisão.

A.2 PROFIBUS

O PROFIBUS é um padrão aberto de rede de comunicação industrial, sendo utilizado em uma ampla variedade de aplicações em automação da manufatura, automação de

processos e automação predial. Sua total independência de fabricantes e sua padronização são garantidas pelas normas EN50170 e EN50254 (ASSOCIAçãO PROFIBUS BRASIL, 2000). Com o PROFIBUS, dispositivos de diferentes fabricantes podem se comunicar sem a necessidade de qualquer adaptação na interface.

O PROFIBUS pode ser usado tanto em aplicações com transmissão de dados em alta velocidade como em tarefas complexas e extensas de comunicação. A organização de usuários PROFIBUS está atualmente trabalhando na implementação de conceitos universais para integração vertical baseada em TCP/IP. O PROFIBUS oferece dois diferentes tipos de protocolos de comunicação: DP (*Decentralized Periphery*) e FMS (*Fieldbus Message Specification*).

O perfil da aplicação define as opções do protocolo e da tecnologia de transmissão requerida nas respectivas áreas de aplicação e para os vários tipos de dispositivos. Este perfil também define o comportamento do dispositivo.

A aplicação de um sistema de comunicação industrial é amplamente influenciada pela escolha do meio de transmissão disponível. Assim sendo, aos requisitos de uso genérico, tais como alta confiabilidade de transmissão, grandes distâncias a serem cobertas e alta velocidade de transmissão, somam-se às exigências específicas da área de automação de processos, tais como operação em área classificada, transmissão de dados e alimentação no mesmo meio físico, entre outros. Partindo-se do princípio de que não é possível atender a todos estes requisitos com um único meio de transmissão, existem atualmente três tipos físicos de comunicação disponíveis no PROFIBUS: RS-485, para uso universal, em especial em sistemas de automação da manufatura; IEC 61158-2, para aplicações em sistemas de automação e controle de processo; Fibra ótica, para aplicações em sistemas que demandam grande imunidade às interferências e grandes distâncias. Atualmente, estão sendo desenvolvidos componentes comerciais com velocidade de 10 e 100 Mbit/s na camada física para PROFIBUS.

A.2.1 Características da rede PROFIBUS

O PROFIBUS especifica as características técnicas e funcionais de um sistema de comunicação industrial, através do qual dispositivos digitais podem se interconectar desde o nível de campo até o nível de células. Por ser um protocolo do tipo multimestre permite a operação conjunta de diversos sistemas de automação, engenharia e visualização com seus respectivos dispositivos periféricos. Os dispositivos mestres também são chamados de estações ativas e os escravos também são chamados estações passivas. A comunicação de dados no barramento é determinada pelos dispositivos mestres. Um mestre pode enviar mensagens sem uma requisição externa, sempre que possuir o direito de acesso ao barramento.

Os escravos são dispositivos de periferia tais como módulos de I/O, válvulas, acionamentos de velocidade variável e transdutores. Eles não têm direito de acesso ao barramento e só podem enviar mensagens ao mestre ou reconhecer mensagens recebidas quando solicitados.

A.2.2 Arquitetura de protocolo

Baseado em padrões reconhecidos internacionalmente, sua arquitetura de protocolo é orientada ao modelo de referência OSI (*Open System Interconnection)* (TANENBAUM, 1996) de acordo com o padrão internacional ISO 7498. Neste modelo a camada 1 (nível físico) define as características físicas de transmissão, a camada 2 (*data link layer*) define o protocolo de acesso ao meio e a camada 7 (*application layer*) define as funções de aplicação (veja Figura A.1). A camada 2 é a camada de enlace de dados, é é composta pelas subcamadas LLC (*Link Logical Control*) e MAC (*Medium Access Control*).

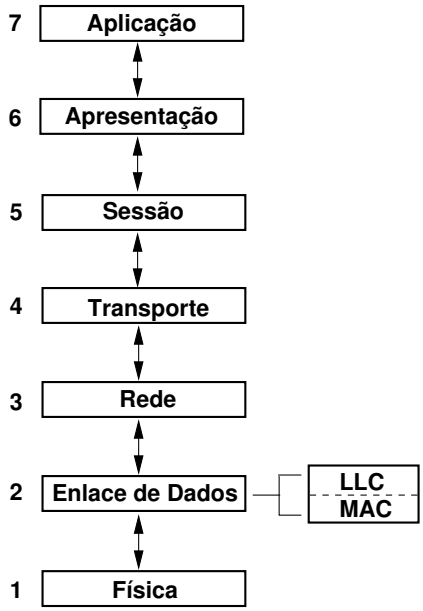

Figura A.1: Modelo de referência OSI

A.2.3 Tendências PROFIBUS

A inovação do PROFIBUS está relacionada ao seu acoplamento transparente com a *Ethernet*. Neste sentido vem-se observando tendências na direção da integração de dados em grandes companhias, desde o sistema de controle até o nível de dispositivos de campo distribuídos.

Os dispositivos PROFIBUS possuem diferentes características de funcionalidade (por exemplo: números de I/O e funções de diagnósticos) ou de parametrização da comuni-

cação, tais como taxa de transmissão e tempo de monitoração. Estes parâmetros variam individualmente para cada tipo de dispositivo e de fabricante e são normalmente documentados nos manuais técnicos. Apesar disto, a fim de torná-los *plug and play*, definiu-se um Arquivo de Dados Eletrônicos do Dispositivo (Arquivo GSD), onde estas informações são armazenadas. Existem diversas ferramentas de configuração para a rede PROFIBUS, contudo, com base nos arquivos GSD, é possível configurar mesmo uma rede PROFIBUS complexa, com os mais diversos dispositivos de diferentes fabricantes, de uma maneira simples, rápida e intuitiva.

Atualmente, o PROFIBUS disponibiliza mais de 2000 produtos e serviços a seus usuários. Além disso, através de um contínuo desenvolvimento tecnológico, novas funções estão sendo disponibilizadas para substituir funcionalidades que anteriormente só podiam ser implementadas em barramentos especiais. Para os usuários isto traz a vantagem de poder usar o PROFIBUS em praticamente todas tarefas de comunicação industrial.

A.3 CAN

Desenvolvida pelas companhias Robert Bosch, a rede CAN (*Controller Area Network*) surgiu em meados dos anos 80 com a finalidade de satisfazer às necessidades crescentes de segurança, conforto e comodidade existentes no mercado automotivo, associadas às exigências governamentais referentes à redução de poluição, consumo e aumento da segurança (LAWRENZ, 1997).

O protocolo CAN, também desenvolvido pela Bosch, foi inicialmente idealizado para uso em automóveis, mas atualmente é encontrado em diversos campos de aplicação, como em sistemas de controle industrial e redes embutidas. O protocolo CAN se tornou um padrão internacional. Está documentado para aplicações em alta velocidade na ISO 11.898 e para aplicações em baixa velocidade na ISO 11.519.

A.3.1 Controle de acesso ao meio

CAN é um protocolo de comunicação serial temporizado guiado a prioridade. Na definição de prioridade é considerado que o identificador de menor valor numérico terá maior prioridade. A rede CAN utiliza o método de acesso ao meio CSMA/DCR (Acesso Múltiplo Sensível à Portadora com Resolução de Colisão Determinística), ou seja, os nós atrasam a transmissão se o barramento estiver ocupado. Quando é detectada a condição de barramento livre qualquer nó pode começar a transmitir. Os conflitos de acesso ao barramento são solucionados por comparação orientada ao *bit*, *bitwise*, dos identificadores das mensagens e funciona da seguinte maneira: enquanto transmite o identificador do objeto de comunicação cada nó monitora o barramento série. Se o *bit* transmitido for recessivo (nível

lógico '1') e for monitorado um *bit* dominante (nível lógico '0'), o nó desiste da transmissão e inicia a recepção dos dados que chegam. O nó que transmite o menor identificador ganha acesso ao barramento e continua a transmissão.

O mecanismo de arbitração é ilustrado na Figura A.2. Observe que até o *bit* 6 todos os nós enviaram o mesmo valor de *bit*. Entretanto, no *bit* 5 o nó 2 transmitiu um *bit* recessivo, perdendo assim a disputa para os nós 1 e 3. No *bit* 2 o nó 1 enviou um *bit* recessivo e entrou em modo de escuta. Dessa forma, o nó 3 ganhou a disputa para acesso ao barramento e seguiu transmitindo a mensagem. Isso significa que a arbitragem é não destrutiva, uma vez que a transmissão do objeto de menor identificador não sofre atraso.

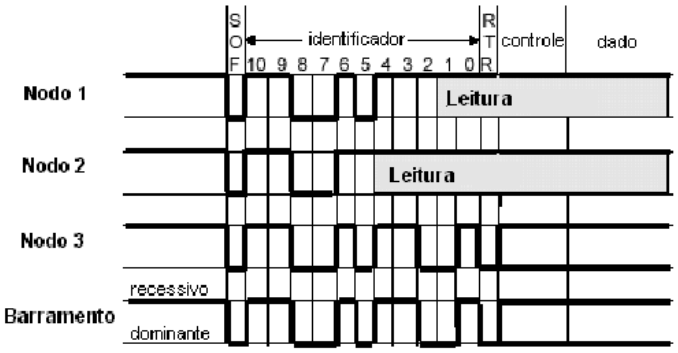

Figura A.2: Processo de arbitração

CAN é usado eficientemente para aplicações de controle em tempo real com um alto nível de segurança. Opera em taxas de até 1 mega *bits* por segundo. Possui excelentes mecanismos de detecção de erros e permite o funcionamento em ambientes austeros.

A.3.2 Aplicações do padrão CAN

Originalmente desenvolvida para uso em carros, atualmente a rede CAN é utilizada em um grande número de aplicações indústrias, incluindo entre outras, aplicações em caminhões, ônibus, agricultura, marítima, médica, construção, automação industrial e têxtil. Em geral o protocolo CAN é adequado para aplicações onde subsistemas baseados em microcontroladores ou dispositivos têm que se comunicar (GMBH, 1991).

Em carros e caminhões as redes CAN são usadas em aplicações que conectam muitas unidades eletrônicas de controle. Essas redes têm baixas taxas (125 $kbit/s$) e são usadas para gerenciamento do motor, controle das unidades de portas, entre outras aplicações. Dentro do veículo, existem ainda, as redes CAN que conectam apenas os dispositivos de entretenimento, como rádios automotivos e aparelhos de CD.

Em transportes públicos, como trens, as redes CAN são usadas para conectar unidades controladoras de freio aos subsistemas. A CiA (*CAN in Automation*) estabeleceu a

CANopen SIG Railways, especificando muitos dispositivos dedicados para trens como: controlador de portas, controlador de freios e portões para os sistemas de barramento de trens. Nestas atividades um grande número de modelos de dispositivos estão envolvidos.

As redes CAN também estão sendo usadas para interligar equipamentos de medição de segurança de tráfego. A vigilância para as luzes e detecção de velocidade realizam diversas funções nos nós separadamente.

A.4 ASI

Promovida pelo governo alemão, a rede ASI (*Actuator-Sensor-Interface*) foi organizada como um projeto coordenado por um consórcio de 11 empresas e outros órgãos. Este projeto foi terminado em 1994 e hoje conta com cerca de 80 membros internacionais e 9 grupos de usuários nacionais, tornando-se o padrão para o nível mais baixo de automação.

A rede ASI é uma rede de automação que permite que atuadores e sensores sejam conectados a um controlador para realizar uma solução de baixo custo e alta eficiência de automação no nível mais baixo de um sistema de automação, onde se encontram os equipamentos mais simples como atuadores e sensores.

A.4.1 Aspectos físicos

A estrutura de uma rede ASI possibilita a redução do cabeamento, uma vez que os sinais e a tensão de alimentação estão em um mesmo cabo, o qual conecta todos os sensores/atuadores com apenas dois fios em um único cabo. Esta simplificação de cabeamento torna a rede ASI mais flexível e mais simples de ser instalada e expandida, tudo isto a um custo menor.

O cabo padrão da rede ASI é o chamado cabo amarelo. Este cabo é do tipo *flat*, não blindado. É revestido com um material que suporta a classe de proteção IP67, mesmo depois de desconectado. O transporte de sinais de dados e tensão de alimentação num mesmo cabo é conseguido com a modulação APM (*Alternate Pulse Modulation*) e com a introdução de uma impedância na fonte de alimentação. Este cabo suporta uma tensão de 24 V DC e até 8 A.

O comprimento máximo desta rede é de 100 metros, extensíveis a até 300 metros com o uso de repetidores/extensores. Não é necessário resistor de terminação.

A.4.2 Controle de acesso ao meio

Do tipo mestre escravo, o controlador ASI sempre é o mestre e sempre tem o controle do barramento, enquanto que os módulos são os escravos e apenas respondem às requisições

dos mestres. Neste princípio, o mestre chama os escravos serialmente, um por vez e ciclicamente até que todo o ciclo seja completado, quando então o procedimento é repetido. Para isto cada escravo recebe um endereço, que é designado automaticamente pelo mestre. Os escravos vêm de fábrica com o endereço 0, que também pode ser atribuído ao escravo através de um endereçador manual.

Pode-se ter até 31 escravos em um barramento, com cada escravo tendo até 4 entradas e 4 saídas digitais, totalizando até 248 entradas e saídas digitais. Adicionalmente, pode-se ter 4 *bits* de parâmetros por escravo, os quais podem ser utilizados para modificar a configuração dos dispositivos conectados ao escravo. Pode-se ter também módulos ativos para a conexão de dispositivos analógicos convencionais.

A rede ASI é uma rede serial e em cada ciclo o mestre percorre todos os escravos conectados a este para monitoramento ou envio de requisições de dados, no caso dos sensores, e de comandos, no caso dos atuadores. Durante o ciclo as falhas são detectadas através de um mecanismo para detecção de falhas. A detecção de falhas também é implementada por *software* no mestre e por um "cão de guarda" no escravo. Este mecanismo permite a diferenciação entre falhas periféricas e falhas de comunicação. No caso de falhas de comunicação um mecanismo de retransmissão está presente.

A freqüência de comunicação com os escravos é de 167 $Kbps$. Uma vez que o tempo para cada escravo é de 150 μs, o tempo máximo de um ciclo é de 5 ms com 31 escravos conectados ao barramento. Obviamente, quanto menor o número de escravos conectados ao barramento menor será o tempo de ciclo. Quanto aos *bits* de parâmetros, estes só poderão ser utilizados uma vez por ciclo. Desta maneira, apenas um escravo pode ser parametrizado a cada ciclo.

A.4.3 ASI 2.1

O texto escrito acima foi baseado na versão 2.0 da especificação ASI. Melhoramentos técnicos, introduzidos na versão 2.1 da especificação ASI, estendem as capacidades da tecnologia ASI e são totalmente compatíveis com a especificação ASI da versão 2.0. A versão 2.1 possibilita até 62 escravos conectados em um mestre, diagnósticos mais detalhados e conexões mais simples a escravos analógicos.

A.5 IEEE 802.3 (Ethernet)

Um dos padrões para redes locais mais utilizados no mundo, a *Ethernet* forma a base do padrão IEEE 802.3 para rede locais, chegando em muitos casos a se usar o termo *Ethernet* para se referir ao padrão IEEE 802.3. O padrão IEEE 802.3 abrange um conjunto de protocolos referentes à camada física e à subcamada MAC do modelo OSI. Com a *Ethernet*

consegue-se transmitir dados à velocidade de 10 $Mbit/s$, 100 $Mbit/s$ (*Ethernet* rápida) e 1 $Gbit/s$ (JR., 2002).

Em razão de sua larga utilização e baixos custos de *hardware*, as redes *Ethernet* tornaram-se bastante atrativas para uso em sistemas de controle em tempo real, embora não tenham sido projetadas para uso em comunicações em tempo real. O empecilho na utilização das redes *Ethernet* em sistemas de controle está no seu mecanismo de controle de acesso ao meio, que não permite prever, nem limitar o tempo que uma estação leva para transmitir.

A.5.1 Controle de acesso ao meio

A *Ethernet* utiliza o protocolo CSMA/CD (Acesso Múltiplo Sensível à Portadora com Detecção de Colisão) para resolver o problema de contenção no meio de comunicação. Assim, quando um nó quer transmitir ele "escuta" a rede. Se a rede está ocupada ele deve esperar até que a rede fique livre. Caso contrário, ele transmite imediatamente. Caso dois ou mais nós detectem que a rede esteja livre e decidam transmitir simultaneamente, as mensagens irão colidir e se corromperão. Contudo, ao transmitir, o nó também escuta a rede a fim de detectar se houve uma colisão. Caso uma colisão seja detectada, o nó aborta a sua transmissão e espera um intervalo de tempo aleatório para retransmitir. Esse intervalo de tempo aleatório é determinado através do algoritmo BEB (*Binary Exponential Backoff*).

Devido ao uso do algoritmo BEB, o atraso em redes *Ethernet* é, em geral, não determinístico. Entretanto, quando a taxa de utilização da rede é baixa, o atraso é praticamente constante, pois o número de colisões é bastante pequeno. Quando a taxa de utilização da rede cresce, o número de colisões também cresce. Conseqüentemente, o atraso aumenta bastante devido ao aumento do número de retransmissões. Além do aumento do atraso, as características não determinísticas do atraso também se acentuam devido ao tempo de espera aleatório para retransmissão (NASCIMENTO, 2004).

A.6 Ethernet sem fio

A inexistência de fios e cabos para conectar os dispositivos, a mobilidade e flexibilidade, proporcionadas pelas redes sem fio, são razões bastante atrativas para a utilização de redes sem fio para fins de controle. Dentre as tecnologias para redes sem fio, as soluções baseadas no padrão IEEE 802.11 (*Wireless Ethernet*) vêm sendo utilizadas para fins de controle. O padrão IEEE 802.11 especifica a camada física e a subcamada MAC da camada de enlace de dados do modelo de referência OSI.

Há duas maneiras possíveis de configurar uma rede sem fio segundo o padrão IEEE 802.11: *ad-hoc* e infraestrutura (*infrastructure*). Na configuração *ad-hoc* a rede não possui uma estrutura fixa e, geralmente, cada nó é capaz de se comunicar com todos os outros nós da rede. Por outro lado, na configuração chamada de infraestrutura existem pontos de acesso a rede que não são fixos, com os quais os nós móveis podem se comunicar (NASCIMENTO, 2004).

A.6.1 Controle de acesso ao meio

O mecanismo de acesso ao meio numa rede *Ethernet* sem fio CSMA/CD não pode ser utilizado, pois não se pode admitir que todas as estações estão ao alcance umas das outras. Para contornar este problema utiliza-se o protocolo CSMA/CA (Acesso Múltiplo Sensível à Portadora com Mecanismo para Evitar Colisão).

Os dispositivos que usam CSMA/CA utilizam um *handshake* de quatro modos, como indicado na Figura A.3, para ganhar acesso ao meio de propagação a fim de assegurar que colisões sejam evitadas. Quando um nó deseja transmitir ele envia um pacote RTS (*Request To Send* - Requisição para transmitir) endereçado ao nó desejado. Se o nó de destino recebe o pacote e está apto a receber o restante da transmissão ele responde com um pacote CTS (*Clear To Send* - Pronto para enviar). O nó que iniciou a transmissão envia então os dados para o nó destino, o qual realiza o reconhecimento de todos os pacotes recebidos retornando um pacote ACK (*Acknowledgement* - Reconhecimento) para cada pacote recebido (NASCIMENTO, 2004).

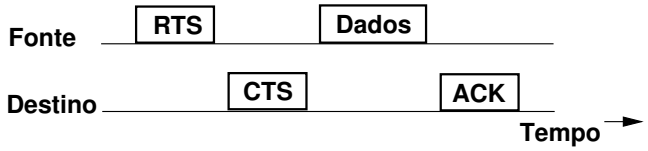

Figura A.3: *Handshake* de quatro modos do protocolo CSMA/CA

A distância entre os nós é um dos fatores que afetam consideravelmente as condições de transmissão numa rede sem fio. Com o aumento da distância entre os nós a razão sinal ruído diminui, o que leva o *hardware* de rede a reduzir a taxa de transmissão. Outros fatores que levam a uma deterioração na transmissão são o ruído do ambiente e o tráfego adicional no canal sem fio.

A.7 ControlNet

Analogamente às redes MAP (*Manufacturing Automation Protocol*) e PROFIBUS, a ControlNet é uma rede de controle que utiliza o mecanismo de passagem de ficha para resolver

o problema de contenção no meio de comunicação. Uma das principais características dessas redes é o seu caráter determinístico, já que o tempo máximo de espera para enviar uma mensagem pode ser limitado pelo tempo de rotação da ficha. No caso da rede ControlNet o mecanismo de passagem da ficha é implícito. Não existe uma ficha que trafega ao longo da rede. Ao invés disso, o protocolo usa uma identificação (um número que varia de 0 a 99) que é atribuída a cada nó para simular o mecanismo de passagem de ficha (NASCIMENTO, 2004).

A.7.1 Controle de acesso ao meio

Da mesma forma que ocorre em redes em que a passagem de ficha é feita de forma explícita, numa rede ControlNet os nós da rede são dispostos logicamente no formato de um anel. Além disso, cada nó conhece o endereço do seu antecessor e de seu sucessor lógico. Ao longo da operação da rede o nó com a ficha transmite quadros de dados até que eles acabem ou que o tempo que o nó deve ficar com a ficha se esgote. O nó então regenera a ficha e a retransmite para o seu sucessor lógico na rede. Se um nó não possui mensagens a serem enviadas ele apenas passa a ficha para o nó sucessor. A localização física do sucessor é irrelevante, uma vez que a ficha é enviada ao vizinho lógico. Como apenas um nó pode transmitir de uma vez, não há colisão de mensagens neste tipo de rede. Seu protocolo também garante um tempo máximo para que cada nó tenha acesso à rede e também oferece mecanismos para regenerar a ficha, caso seu possuidor pare de transmitir e não a passe adiante (NASCIMENTO, 2004).

A.7.2 Características de atraso

A rede ControlNet tem características determinísticas. Cada nó só pode transmitir por um tempo limitado enquanto estiver com a ficha. Assim, conhecendo-se o período que cada nó transmite, o tempo que cada nó permanece com a ficha e a quantidade de nós na rede, determina-se com exatidão o tempo que um nó levará para ter acesso ao meio de transmissão. Caso essas informações não sejam conhecidas, no mínimo é conhecido o tempo máximo de acesso ao meio que é caracterizado pela rotação da ficha no anel lógico (NASCIMENTO, 2004).

A.8 DeviceNet

De propriedade da Allen-Bradley até 1992, a DeviceNet é uma tecnologia de comunicação aberta e de baixo custo. Desde sua criação, a Allen-Bradley começou a dividir informações e convidar não apenas parceiros estratégicos, mas competidores diretos para se tornarem

membros da DeviceNet. Em março de 1994 foi liberada no ICEE. Em 1995 a Allen-Bradley volta a DeviceNet para a ODVA (*Open DeviceNet Vendors Association*).

Baseada no padrão OSI, utilizado na camada de enlace de dados do protocolo CAN, a DeviceNet é usada nas indústrias automotivas, de manufaturas, agrícolas, médicas, de construção civil, marítimas, espaciais e outras.

O *link* de comunicação DeviceNet é baseado em radiodifusão fechada e protocolos de comunicação. É usada para interligar, através de uma rede simples, sensores e atuadores e uma série de outros dispositivos industriais, tais como: chaves limitadoras, válvulas *manifolds*, sensores de proximidade (indutivos, capacitivos, fotoelétricos), motores de partida, sensores de processos, leitores de código de barras, variadores de freqüência, painéis de *displays* e operadores de interfaces.

O fato de ser uma rede simples elimina instalações elétricas extensivas e falhas devido ao grande número de conexões, reduzindo os custos e o tempo de comunicação entre os dispositivos instalados, com uma condição confiável de intercambialidade entre os instrumentos de diversos fabricantes. A conexão direta influencia a comunicação entre os dispositivos, implementando níveis de autodiagnóstico que nem sempre são facilmente acessíveis ou disponíveis através das interfaces de I/O.

As topologias em linha e árvore podem ser aplicadas, sendo que o comprimento máximo varia de acordo com o tipo de cabo e a taxa de transmissão da rede, restringindo-se também o tamanho das derivações. Permite o endereçamento de até 64 nós MAC ID (*Media Access Control Identifiers*).

Utiliza um cabo padrão de 2 pares trançados, sendo um dos pares responsável pela distribuição da tensão de alimentação de 24 V DC nos diversos n´s, e o outro utilizado para o sinal de comunicação.

A.8.1 Controle de acesso ao meio

A topologia de rede é reta, com restritas derivações e utiliza o método de acesso ao meio CSMA/BA (*Carrier Sense Multiple Acess com Bitwise Arbitration*), suportando múltiplos modelos de redes como: mestre escravo, múltiplos mestres, e ponto a ponto (introduzido em 1996 pelo Grupo Sistema SIG da ODVA para aprovação, sendo adaptada, revisada e implementada em 1996 e 1997) (NASCIMENTO, 2004).

A.8.2 Transmissão de mensagens

O sinal é transmitido através de uma onda digital quadrada com a codificação NRZ (*Non Return to Zero*). Na saída de um nó tem-se 64 *bits* padrões. Destes, 24 *bits* são destinados às transferências de mensagens explícitas. Também são suportadas mensagens

fragmentadas (NASCIMENTO, 2004).

O processo define dois estados lógicos, chamados de dominante (0) e recessivo (1). A rede permanece no modo recessivo somente enquanto não há nenhuma transmissão. Se a rede não estiver ocupada, qualquer participante pode tentar transmitir informações.

A.8.3 Sincronização

A DeviceNet é baseada no barramento CAN que utiliza 11 *bits* de identificação padrão, dos quais 5 *bits* são para o sinal de mensagem e 6 *bits* são para o MAC ID (64 endereços).

A.9 Bluetooth

O *Bluetooth* surgiu em 1998 quando a Ericsson, a Nokia, a IBM, a Intel e a Toshiba formaram o consórcio denominado *Bluetooth* SIG (*Special Interest Group*) com o objetivo de expandir e promover o conceito *Bluetooth* e estabelecer um novo padrão industrial (HAARTSEN, 2000). Este consórcio cresceu rapidamente com o suporte de companhias líderes em telecomunicações, eletrodomésticos e PCs, interessadas no desenvolvimento de produtos baseados na nova especificação. Atualmente, cerca de 1200 empresas participam deste consórcio em todo o mundo.

O *Bluetooth* é um padrão para comunicação sem-fio, sendo realizado por meio de conexões de rádio *ad hoc* (ROYER; TOH, 1999) de curto alcance e baixo custo. Através do *Bluetooth*, usuários podem conectar uma ampla variedade de dispositivos de computação, de telecomunicações e eletrodomésticos de uma forma bastante simples, sem a necessidade de adquirir, carregar ou conectar cabos de ligação. A interligação desses dispositivos deve ocorrer de uma forma automática e sem que o usuário necessite se preocupar com isso.

De uma forma geral, o padrão Bluetooth visa facilitar as transmissões em tempo real de voz e dados, permitindo conectar quaisquer aparelhos eletrônicos, fixos ou móveis, que estejam de acordo com a tecnologia.

A.9.1 Redes no bluetooth

Os dispositivos *Bluetooth* se comunicam entre si e formam uma rede denominada *piconet*, na qual podem existir até 8 dispositivos interligados, sendo um deles o mestre e os outros escravos. Nas aplicações *Bluetooth*, várias *piconets* independentes e não-sincronizadas podem se sobrepor ou existir na mesma área. Neste caso, forma-se um sistema *ad hoc* (ROYER; TOH, 1999) disperso denominado *scatternet*, composto de múltiplas redes, cada uma contendo um número limitado de dispositivos. Na Figura A.4 são apresentadas estas idéias.

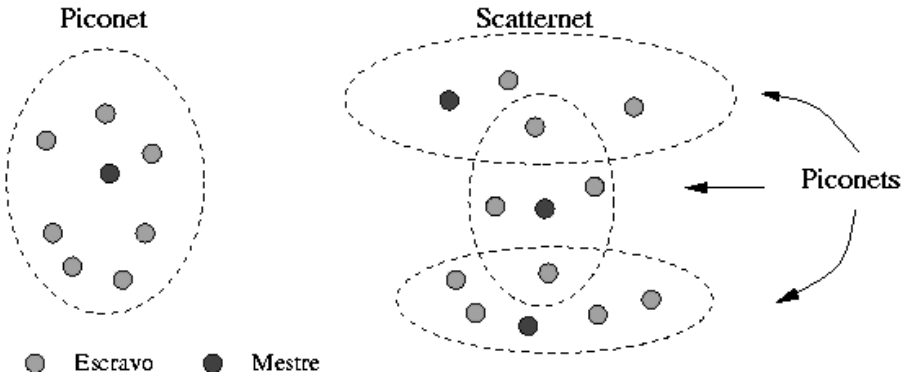

Figura A.4: Redes formadas entre dispositivos Bluetooth

Dispositivos *Bluetooth* operam na faixa ISM (*Industrial, Scientific, Medical*), centrada em 2,45 GHz. Em países onde a faixa ISM é diferente, são necessários alguns ajustes, pois a largura de banda e a localização da faixa ISM diferem. A França é um desses países.

A.9.2 Protocolo de acesso múltiplo

A comunicação entre os dispositivos *Bluetooth* é realizada através do estabelecimento de um canal FH-CDMA (*Frequency Hopping - Code-Division Multiple Access*). Nesta técnica, o transmissor envia um sinal sobre uma série aparentemente aleatória de freqüências de rádio. Um receptor, "saltando" entre tais freqüências e em sincronia com o transmissor, capta o sinal. A mensagem é totalmente recebida apenas se o receptor conhecer a série de freqüências na qual o transmissor "saltará" para enviar o sinal.

Para a operação do *Bluetooth*, na faixa ISM de 2,45 GHz, foram definidas 79 portadoras espaçadas de 1 MHz. Portanto, existem 79 freqüências nas quais instantaneamente um dispositivo pode estar transmitindo. Um grande número de seqüências pseudo-aleatórias de freqüências foi definido. A seqüência particular de freqüências de um canal é estabelecida pelo dispositivo mestre da *piconet*, que é o responsável pelo controle do canal. Todos os outros dispositivos da *piconet* são escravos e devem se sincronizar ao mestre. O dispositivo mestre muda sua freqüência de transmissão 1600 vezes por segundo objetivando minimizar potenciais interferências.

A.9.3 Controle de acesso ao meio

O *Bluetooth* foi otimizado para permitir que um número elevado de comunicações descoordenadas ocorra dentro da mesma área. De modo diferente de outras soluções *ad hoc* (ROYER; TOH, 1999), onde todos os dispositivos compartilham o mesmo canal, no *Bluetooth* existe um grande número de canais independentes e não-sincronizados, cada qual servindo apenas um número limitado de participantes.

Cada canal está associado a uma *piconet* e é identificado pela seqüência de freqüências e pelo relógio (fase do salto) do dispositivo mestre. Todo o controle de tráfego dentro da *piconet* é realizado pelo dispositivo mestre. Somente comunicações ponto-a-ponto entre o dispositivo mestre e um escravo ou comunicações ponto-a-multiponto entre o dispositivo mestre e os escravos são possíveis.

Para evitar a colisão devido a múltiplas transmissões de dispositivos escravos, o mestre utiliza a técnica de nomeação (*polling*). Assim, apenas o dispositivo indicado no *slot* mestre-para-escravo pode transmitir no *slot* escravo-para-mestre seguinte.

A.9.4 Tipos de serviços

Os enlaces sem-fio no *Bluetooth* suportam, tanto serviços síncronos para tráfego de voz, quanto, serviços assíncronos para transmissão de dados. Em um enlace assíncrono, a taxa máxima que um usuário pode obter é de 723,2 kbps. No sentido contrário, a taxa máxima é de 57,6 kbps. Os serviços síncronos são prioritários e têm enlace *full-duplex* com taxa máxima de 64 kbps em ambas as direções.

Referências Bibliográficas

ANALOG DEVICES. *MicroConverter Technical Note - uC003 - The ADuC812 as a 1451.2 STIM*. Ver 1.0 sept. 1999. [S.l.], 1999.

ANALOG DEVICES. *MicroConverter Technical Note - uC002 Developing in C with the Keil uVision2 IDE*. Ver 2.0. [S.l.], 2001.

ANALOG DEVICES. *ADuC832, MicroConverter, 12-Bit ADCs and DACs with Embedded 62 kBytes Flash MCU*. Inc., 2002. [S.l.], 2002.

ANALOG DEVICES. *ADuC8xx SAR Evaluation Board Reference Guide, MicroConverter QuickStart^{TM} Development System*. Version a.3. [S.l.], 2002.

ASSOCIAçãO PROFIBUS BRASIL. *PROFIBUS - Descrição Técnica*. 4.002b. ed. [S.l.], 2000.

BARFORD, L.; LI, Q. Choosing designs of calibration transducer eletronic data sheet. *IEEE*, v. 0-7803-5276-9/99, p. 320–325, 1999.

BURCH, J.; EIDSON, J.; HAMILTON, B. The design of distributed measurement systems based on the ieee1451 standards and distributed time services. *IEEE*, v. 0-7803-5890-2/00, p. 529–534, 2000.

CHOW, M.-Y.; TIPSUWAN, Y. Network-based control systems: A tutorial. *IECON'01: The 27th Annual Conference of the IEEE Industrial Electronics Society*, v. 0-7803-7108-9/01/(C)2001 IEEE, p. 1593–1602, 2001.

COSTA, B. *Wireless Distributed Sensing and Actuation*. Dissertação — University of Wollongong, October 2002.

DALLAS. *O TINI Specification and Developer's Guide*. First. [S.l.], 2001.

DALLAS. *Documentation TINI Firmware 1.02f, C.txt (DC, 22.03.02)*. [S.l.], 2002.

DALLAS. *Documentation TINI Firmware 1.02f, Native_ API.txt (DC, 22.03.02)*. [S.l.], 2002.

DALLAS. *Documentation TINI Firmware 1.02f, Native_ Methods.txt (DC, 22.03.02).* [S.l.], 2002.

DALLAS. *Documentation TINI Firmware 1.02f, Native_ ReadMeNOW.txt (DC, 22.03.02).* [S.l.], 2002.

DALLAS. *Documentation TINI Firmware 1.02f, README.txt (DC, 22.03.02).* [S.l.], 2002.

DALLAS. *Documentation TINI Firmware 1.02f, Running_ JavaKit.txt (DC, 22.03.02).* [S.l.], 2002.

DALLAS. *Documentation TINI Firmware 1.02f, slush.txt (DC, 22.03.02).* [S.l.], 2002.

DALLAS. *Documentation TINI Firmware 1.11, Serial_ Capibilities.txt (DC, 03.22.02).* [S.l.], 2002.

DALLAS. *Documentation TINI Firmware 1.02f, BuildDependency_ README.txt (DC, 15.05.03).* [S.l.], 2003.

DALLAS. *Documentation TINI Firmware 1.02f, Building_ Applications.txt (DC, 15.05.03).* [S.l.], 2003.

DALLAS. *Documentation TINI Firmware 1.11, Installation.txt (DC, 01.24.03).* [S.l.], 2003.

ESTL, H. *Application Note, SPI interface and use in a daisy-chain bus configuration.* V 1.2. [S.l.], 2002.

GMBH, R. B. *CAN Especification Version 2.0.* Stuttgart: [s.n.], 1991.

GOMAA, H. *Designing Concurrent, Distributed, and Real-Time Applications with UML.* First. USA and Canada: Addison-Wesley, 2000.

HAARTSEN, J. C. The bluetooth radio system. *IEEE Personal Communications*, v. 1, p. 28–36, 2000.

IEEE. Standard for a smart transducer interface for sensors and actuators - transducer to micro-processor communication protocols and transducer eletronic data sheet (teds) formats. *IEEE*, IEEE Std 1451.2-1997, p. 1–120, 1997.

IEEE. Standard for a smart transducer interface for sensors and actuators - network capable application processor (ncap) information model. *IEEE*, IEEE Std 1451.1-1999, 1999.

IEEE. Standard for a smart transducer interface for sensors and actuators - digital communication and transducer electronic data sheet (teds) formats for distributed multidrop systems. *IEEE*, IEEE Std 1451.3-2003, p. 1–185, 2004.

IEEE. Standard for a smart transducer interface for sensors and actuators - mixed-mode communication protocols and transducer electronic data sheet (teds) formats. *IEEE*, IEEE Std 1451.4-2004, p. 1–439, 2004.

IEEE. *IEEE-P1451 Draft Standard for Smart Transducer Interface for Sensors and Actuators - A high-speed CANopen based Transducer Network Interface for Intrinsically Safe and Intrinsically Safe applications - http://grouper.ieee.org/groups/1451/6/TermsDefinitions.htm.* May, 2005. [S.l.], 2005.

JOHNSON, R. N. Building plug-and-play networked smart transducers. *Eletronics Development Corporation*, October, p. 1–18, 1997.

JOHNSON, R. N.; WOODS, S. P. Overview and status update for ieee 1451.2: Transducer to microprocessor communications protocols and transducer electronic data sheet (teds) formats. *Telemonitor, Inc and Agilent Technologies*, October 1998, p. 1–8, 1998.

JR., B. Z. D. e N. A. Evolução do padrão ethernet. *CBPF*, CBPF-NT-002/02, p. 1–15, 2002.

KEIL SOFTWARE. *Cx51 Compiler - Optimizing C Compiler and Library Reference for Classic and Extended 8051 Microcontrollers.* User's guide 09.2001. [S.l.], 2001.

LAWRENZ, W. *CAN System Engineering: From Theory to Practical Applications.* Hardcover: ISBN 0-387-94939-9, 1997.

LEE, K. A synopsis of the ieee p1451 - standards for smart transducers communication. *NIST*, Maryland, USA, p. 1–6, 1999.

LEE, K.; CHEN, S. C. A mixed-mode smart transducer interface for sensors and actuators. *Sound and Vibration*, April 1998, p. 2–5, 1998.

LEE, K.; SCHNEEMAN, R. D. A standardized approach for transducer intefacing: Implementing ieee-p1451 smart transducers interface draft standards. *United Stades Department of Commerce, NIST, Manufacturing Engineering Laboratory, Automated Production Technology Division*, p. 1–15, 1996.

LEE, K. B.; SCHNEEMAN, R. D. Distributed measurement and control based on the ieee 1451 smart transducer interface standard. *IEEE Transactions on Instrumentation and Measurement*, v. 49, no. 3, June 2000, p. 621–627, 2000.

LISBôA, H. M. D. e P. J. D. T. C. A. L. *Java Como Programar*. Quarta. Porto Alegre, RS: Artmed Editora S.A., 2003.

LIU, J. W. S. *Real-time Systems*. 1º edition. ed. New Jersey: Prentice-Hall, 2000.

MANDERS, E.-J.; BARFORD, L. A. Diagnosis of a continous dynamic system from distributed measurements). *IEEE*, v. 0-7803-5890-2/00, p. 546–551, 2000.

MARTINEZ, C. *Application Note AN 113, Serial Memory Interface: The benefits of SPI over I 2 C and μ Wire*. October 1997. [S.l.], 1997.

MAXIM/DALLAS. *Application Note, Designing IP sensors using TINI*. 12/31/2001. ed. [S.l.], 2001.

MAXIM/DALLAS. *Application Note 148, Guidelines for Reliable 1-Wire Networks*. 061902. ed. [S.l.], 2002.

MAXIM/DALLAS. *Application Note 155, 1-Wire Software Resource Guide Device Description*. 052902. ed. [S.l.], 2002.

MAXIM/DALLAS. *Application Note 196, Designing a Virtual Modem using TINI*. 04/08/2002. ed. [S.l.], 2002.

MAXIM/DALLAS. *Application Note 198, Networked Temperature Monitoring*. 052902. ed. [S.l.], 2002.

MAXIM/DALLAS. *Application Note 702, Using TINI Point-to-Point Protocol (PPP)*. 053002. ed. [S.l.], 2002.

MAXIM/DALLAS. *Application Note 704, Asynchronous Serial-to-Ethernet Device Servers*. 100102. ed. [S.l.], 2002.

MAXIM/DALLAS. *DS80C390 Dual CAN High-Speed Microprocessor*. Rev: 032904. [S.l.], 2002.

MAXIM/DALLAS. *Application Note 195, DSTINI1 (TINIm390) Verification Module Chipset Reference Design*. Rev: 051403. [S.l.], 2003.

MAXIM/DALLAS. *Application Note 708, Exploring Tiny InterNet Interface (TINI)*. Rev: 042903. [S.l.], 2003.

MAXIM/DALLAS. *High-Speed Microcontroller User's Guide: DS80C390 Supplement.* Rev: 111103. [S.l.], 2003.

MOORE, J. O.; ROSS, T.; JOHNSON, R. N. Migrating legacy applications to the ieee 1451 standards. *Technical Paper for Sensors Expo*, May 1999, p. 1–7, 1999.

MOTOROLA. *SPI Block User Guide, Original Release Date: 21 JAN 2000, Revised: 06 Mar 2002.* V02.06. [S.l.], 2002.

MOZEK, M. et al. Calibration and error correction algorithms for smart pressure sensors. *IEEE*, v. 0-7803-7527-0/02, p. 240–243, 2002.

NASCIMENTO, E. J. do. *Análise da Estabilidade e do Desempenho de Sistemas de Controle via Redes de Comunicações com Atraso Aleatório.* Dissertação — UFCG, Novembro 2004.

NATIONAL SEMICONDUCTOR. *Precision Centigrade Temperature Sensors.* Ds005516, november 2000. [S.l.], 2000.

NIST. *IEEE-P1451 Draft Standard for Smart Transducer Interface for Sensors and Actuators - http://ieee1451.nist.gov/.* May, 2005. [S.l.], 2005.

NöLKER, N.; KLEMENZ, A. *Interfacing Slow Peripherals to D.Module via SPI, Application Note.* D.signt 1999. [S.l.], 1999.

PASCHOA, M. P.-J. T. C. R. *Fundamentos do Desenho Orientado a Objeto com UML.* São Paulo, SP: MAKRON Books, 2001.

POOLEY, P. S. e R. *Using UML Software Engineering with Objects and Components.* [S.l.]: Addison-Wesley, 2000.

ROYER, E. M.; TOH, C. A review of current routing protocols for ad hoc mobile wireless networks. *IEEE Personal Communications*, p. 46–55, 1999.

SCHNEEMAN, R. D. Implementing a standards-based distributed measurement and control application on the internet). *United Stades Department of Commerce, NIST, Manufacturing Engineering Laboratory, Automated Production Technology Division*, June 1999, p. 1–34, 1999.

SIEMENS. *Application Note, Triple-Halfbridge Driver with Serial Peripheral Interface I/O.* Tle 6208-3 g, 1998-04-01. [S.l.], 1998.

STALLINGS, W. *Operating Systems: Internals and Design Principles.* Third. New Jersey: Prentice-Hall, 1999.

STMICROELECTRONICS. *Application Note, SPI communication between ST7 and EEPROM*. An970/1098. [S.l.], 1999.

SUN. *Java Native Interface, http://java.sun.com/docs/books/tutorial/native1.1/concepts/index.html*. June, 2004. [S.l.], 2004.

SVEDA, M.; VRBA, R. An integrated framework for internet-based applications of smart sensors. *IEEE*, v. 0-7803-7454-1/02, p. 1543–1548, 2002.

SVEDA, M.; VRBA, R. An integrated framework for sensor-based embedded systems. *Proceedings of the Ninth Annaul IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'02)*, v. 0-7695-1549-5/02, p. 195–202, 2002.

SYSTRONIX. *JStamp^{TM} Real-Time Native Java^{TM} Network Module - www.systronix.com*. [S.l.], 2004.

SYSTRONIX. *JStik^{TM} Comparison RealTime Java for the real world - http://www.jstik.com/compare.htm*. [S.l.], 2005.

TANENBAUM, A. S. *Computer Networks*. Third. ISBN 0-13-349945-6: Prentice-Hall, 1996.

TEXAS INSTRUMENTS. *TMS320F28x DSP Serial Peripheral Interface (SPI) Reference Guide*. Spru059a june 2003. [S.l.], 2003.

UBICOM. *Application Note, Serial Peripheral Interface (SPI) and Microwire/Plus implementation Using the SX Communications Controller*. November 2000. [S.l.], 2000.

WALL, R. W.; EKPRUKE, A. Developing an ieee 1451.2 compliant sensor for real-time distributed measurement and control in an autonomous log skidder. *Proceedings of The 29th Annual Conference of the IEEE Industrial Electronics Society Paper*, November 2003, p. 1–6, 2003.

WOLF, W. *Computers as Components. Principles of Embedded Computing System Design*. San Francisco, CA: MK, 2001.

WOODS, S. P. et al. Ieee 1451.2 smart transducer interface module. *Proceedings of SENSORS Conference, Philadelphia, PA; 1996 (October 22-24)*, 1996.