



UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

“Desenvolvimento de um Nó de Rede com Diferentes Interfaces de Acordo com o Padrão IEEE 1451 Utilizando o Processador Nios II e o Sistema Operacional Embarcado μ Clinux”

TÉRCIO ALBERTO DOS SANTOS FILHO

Orientador: Prof. Dr. Alexandre César Rodrigues da Silva

Tese apresentada à Faculdade de Engenharia
- UNESP – Campus de Ilha Solteira, para
obtenção do título de Doutor em Engenharia
Elétrica.

Área de Conhecimento: Automação.

Ilha Solteira - SP

Maio de 2012

FICHA CATALOGRÁFICA

Desenvolvido pela Seção Técnica de Aquisição e Tratamento da Informação

S237d Santos Filho, Tércio Alberto dos.
Desenvolvimento de um nó de rede com diferentes interfaces de acordo com o padrão ieee 1451 utilizando o processador nios ii e o sistema operacional embarcado uclinux / Tércio Alberto dos Santos Filho. -- Ilha Solteira: [s.n.], 2012
182 f. : il.

Tese (doutorado) - Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira. Área de conhecimento: Automação, 2012

Orientador: Alexandre César Rodrigues da Silva
Inclui bibliografia

1. Redes de transdutores inteligentes. 2. Sistemas embarcados. 3. Padrão IEEE 1451. 4. Sistema operacional embarcado. 5. Rede de sensores sem fio Zigbee. 6. Rede de sensores cabeada.

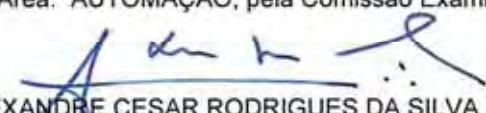
CERTIFICADO DE APROVAÇÃO

TÍTULO: Desenvolvimento de um Nó de Rede com Diferentes Interfaces de Acordo com Padrão IEEE 1451 Utilizando o Processador Nios II e o Sistema Operacional Embarcado uClinux

AUTOR: TÉRCIO ALBERTO DOS SANTOS FILHO

ORIENTADOR: Prof. Dr. ALEXANDRE CESAR RODRIGUES DA SILVA


Aprovado como parte das exigências para obtenção do Título de DOUTOR EM ENGENHARIA ELÉTRICA, Área: AUTOMAÇÃO, pela Comissão Examinadora:


Prof. Dr. ALEXANDRE CESAR RODRIGUES DA SILVA
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira


Prof. Dr. NOBUO OKI
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira


Prof. Dr. APARECIDO AUGUSTO DE CARVALHO
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira


Prof. Dr. EDSON ANTONIO BATISTA
Departamento de Engenharia Elétrica / Universidade Federal de Mato Grosso do Sul


Prof. Dr. ELNATAN CHAGAS FERREIRA
Departamento Demic / Universidade Estadual de Campinas

Data da realização: 25 de maio de 2012.

Aos meus pais, Tercio e Eridam,
à meu irmão Carlos e a minha
querida esposa Denise.

DEDICO

Agradecimentos

Agradeço a Deus por ter dado forças em todos os momentos de minha vida.

Agradeço aos meus pais Tércio Alberto dos Santos e Eridam Pereira da Silva Santos, por toda a dedicação na minha formação como pessoa e profissional, pelo carinho, paciência e amor. Ao meu irmão, Carlos Eduardo da Silva Santos, que sempre me apoiou nos diversos momentos da minha vida, pelos conselhos, pela amizade e confiança.

Agradeço em especial a minha querida esposa, Denise Rodrigues Dias dos Santos que durante todo o período dos meus estudos, me apoiou, deu forças, incentivou nas minhas escolhas e pelo amor.

Agradeço ao meu professor, orientador e amigo Alexandre César Rodrigues da Silva por ter me ensinado e pelo apoio nessa fase de minha vida contribuindo não só como orientador, mais sendo um grande amigo também.

Agradeço aos meus avós, Açodio e Isaura.

Agradeço ao meu sogro e minha sogra, José Rodrigues Dias e Alice Rodrigues Dias, pela ajuda, confiança e pelo apoio.

Agradeço a minha cunhada Mirelle e minha sobrinha Lavínia.

Agradeço as minhas cunhadas, concunhados, sobrinhos e sobrinhas Daniela, Joyce, Denildo, Valdir, João Vitor, Guilherme, Gabriela, Maria Heloisa e Manuela.

Aos meus familiares: Olivar, Patricia, André, Leandro, Olívio, Carmen Luzia, Marcelo Azevedo, Lara, Elma, Ernesta, Iara, Maria do Rosário, Agustinho, Neide, Keila, Flávio, Marina, minha madrinha Cristiane, Luciano e Pedro Henrique.

Aos meus amigos de infância, Rafael Ratke e sua esposa Bruna, Rodrigo Mendonça de Souza e noiva Jaína, Vinícius da Cunha e Heverton Barros de Macêdo e sua esposa Isabelle.

Aos meus amigos Marcos Antônio Estremote e esposa Elaine, Tiago da Silva Almeida, Marcelo Estremote e noiva Andressa, Rafael (Ilha Solteira), Marcelo Sanches, Mateus, Marcos Vinícius e Alex.

Agradeço aos meus amigos que estiveram comigo durante minhas pesquisas realizadas no laboratório dividindo minhas alegrias e conquistas.

Aos meus professores de pós-graduação e técnicos de laboratório.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), processos: (140261/2008-7) e (307255/2009-3). Ao Programa de Pós-Graduação em Engenharia Elétrica - PPGEE, Faculdade de Engenharia Elétrica de Ilha Solteira - FEIS.

Resumo

Os sistemas operacionais possuem um papel fundamental nos microcomputadores, realizando o interfaceamento entre os aplicativos e o *hardware*. Além dos microcomputadores, atualmente, os sistemas operacionais são empregados em ambiente com arquitetura restrita, como os microcontroladores, denominando-os como sistemas operacionais embarcados. Algumas das aplicações para os sistemas operacionais embarcados são voltadas para as redes de transdutores inteligentes, realizando o gerenciamento e controle do sistema em que atua. Neste trabalho apresenta-se o desenvolvimento de um nó de rede denominado NCAP, utilizando o kit DE2 e o sistema operacional embarcado μ Clinux para o monitoramento e controle dos TIMs com diferentes interfaces conforme a norma IEEE 1451. O NCAP desenvolvido possui como característica a conexão de diferentes TIMs com recursos de controle e monitoramento com distintas interfaces de comunicação, com fio (RS-232) e sem fio (ZigBee). Para realização dos testes, foram desenvolvidos 4 TIMs com diferentes características, sendo: 2 STIMs (IEEE P1451.2) e 2 WTIMs (IEEE 1451.5). Os testes foram realizados utilizando uma interface de rede externa, a Ethernet, que, por meio do microcomputador, é possível acessar a página web em um servidor embarcado instalado no nó de rede NCAP. Com isso, é possível realizar o controle e o monitoramento dos transdutores conectados ao TIM independente da interface de comunicação sem fio ou com fio. Além do acesso aos transdutores, o nó possui características que facilitam o gerenciamento, como: servidor de FTP e acesso remoto. O desenvolvimento do nó de rede embarcado, padronizado pela norma IEEE 1451, possibilita maior flexibilidade de implantação em locais remotos e facilita a ampliação do sistema sem efetuar grandes modificações na rede. Uma característica importante é a dinâmica de desenvolvimento em relação ao *hardware* implementado no FPGA com o processador Nios II, sendo possível desenvolver sistemas utilizando apenas os recursos necessários para cada aplicação. Além das características de *hardware*, outro aspecto foi a implementação do sistema operacional embarcado em uma arquitetura dinâmica, disponibilizando uma flexibilidade ao sistema, ampliando a área de pesquisa e apresentando uma nova metodologia de desenvolvimento do padrão IEEE 1451.

Palavras chave: IEEE 1451. Interfaces. NCAP. Nios II. RS-232. Sistema operacional embarcado. TIM. Transdutores inteligentes, μ Clinux, ZigBee.

Abstract

The operating systems have a key role in microcomputers, performing interfacing between applications and the hardware. In addition to computers, currently embedded operating systems are employed in an environment with restricted architecture, such as microcontrollers, terming them as embedded operating systems. Some of the applications for embedded operating systems are geared for smart transducer networks, making the management and control system in which it operates. This work presents the development of a network node denominated NCAP using the DE2 kit and embedded operating system μ Clinux for monitoring and control of TIMs with different interfaces based on IEEE 1451. The NCAP has developed the characteristic connection of different TIMs with monitoring and control capabilities with different communication interfaces, wired (RS-232) and wireless (ZigBee). The testing, four TIMs were developed with different characteristics, as follows: 2 STIMs (IEEE P1451.2) and two WTIMs (IEEE 1451.5). The tests were performed using an external network interface, Ethernet, that through the PC, you can access the web page in an embedded server installed on the network node NCAP. This makes it possible to perform control and monitoring of transducers connected to the TIM interface regardless of wireless or wired. In addition to access to the transducers, the node has features that facilitate the management, such as FTP server and remote access. The development of embedded network node, standardized by IEEE 1451, allows for greater flexibility of deployment in remote locations and facilitates the expansion of the system without making major changes in the network. An important feature is the dynamic development in relation to the hardware implemented in FPGA with Nios II processor, it is possible to develop systems using only the resources required for each application. Another important work was the implementation of embedded operating system in a dynamic architecture, providing flexibility to the system, expanding the search area and presenting a new methodology for the development of IEEE 1451.

Keywords: Embedded operating system. IEEE 1451. Interfaces. NCAP. Nios II. RS-232. Smart transducers. TIM. μ Clinux. ZigBee.

Lista de Figuras

1	Evolução das redes de transdutores inteligentes.	29
2	Diagrama da família do padrão IEEE 1451.	47
3	Bloco NCAP genérico com as principais descrições.	49
4	Hierarquia de classes do padrão IEEE 1451.	50
5	Modelo do padrão IEEE P1451.2.	52
6	Modelo do padrão IEEE 1451.3.	53
7	Modelo do padrão IEEE 1451.4.	53
8	Camada de enlace e camada física.	54
9	Modelo do padrão IEEE 1451.5.	55
10	Modelo do padrão IEEE P1451.6.	56
11	Modelo do padrão IEEE 1451.7.	56
12	Transmissão serial síncrona.	59
13	Transmissão serial assíncrona.	59
14	Nível de tensão utilizado pelo padrão RS-232.	61
15	Mestre/escravo utilizando barramento SPI.	62
16	Camadas utilizadas pelo padrão ZigBee baseado no modelo OSI.	63
17	Topologia estrela utilizada nas redes ZigBee.	65
18	Topologia <i>mesh</i> utilizada nas redes sem fio ZigBee.	66
19	Topologia estrela utilizada nas redes Ethernet.	67

20	Quadro Ethernet.	67
21	Interface do ambiente SoPC Builder.	72
22	Exemplo de código em Verilog no ambiente Quartus II.	74
23	Definição dos processadores no ambiente de desenvolvimento SoPC Builder.	75
24	Etapas de desenvolvimento em ambiente EDA.	78
25	Kit DE1 Altera.	78
26	Kit DE2 Altera.	79
27	Mecanismo de execução do CGI.	80
28	Ambiente de desenvolvimento Nios II.	82
29	Exemplo de código em C para acesso aos periféricos no ambiente μ Clinux.	83
30	Exemplo de TIM baseado no padrão IEEE P1451.2.	85
31	Diagrama de estado de operação do TIM.	86
32	Diagrama de estado de operação do TransducerChannel.	86
33	Diagrama de bloco do STIM_1.	87
34	STIM_1 desenvolvido em laboratório.	88
35	Diagrama de bloco do STIM_2.	88
36	STIM_2 desenvolvido em laboratório.	89
37	Diagrama de bloco do WTIM_1.	90
38	WTIM_1 desenvolvido em laboratório.	90
39	Diagrama de bloco do WTIM_2.	91
40	WTIM_2 desenvolvido em laboratório.	91
41	Protocolo de requisição de controle do motor de passo.	92
42	Protocolo de resposta de controle do motor de passo.	92
43	Protocolo de requisição da temperatura.	92
44	Protocolo de resposta da temperatura referente ao WTIM_1.	93
45	Protocolo de requisição de controle do motor CC.	93
46	Protocolo de resposta do controle do motor CC referente ao WTIM_1.	93

47	Sistema baseado na estrutura de camadas do modelo OSI.	96
48	Esquemático do sistema implementado em laboratório.	96
49	Estrutura para o desenvolvimento do <i>hardware</i> do nó de rede IEEE 1451.1.	98
50	Ambiente SoPC Builder com os módulos definidos para o desenvolvimento do nó de rede embarcado.	99
51	Interface de configuração do módulo UART.	100
52	Mapa do endereçamento do projeto.	100
53	Bloco dos módulos.	101
54	Ambiente Quartus II com código verilog.	102
55	Configurações das variáveis dos pinos de entrada e saída da interface UART.	103
56	Funções de entrada e saída da interface UART.	103
57	Funções das interfaces SPI e Ethernet.	103
58	Configurações das variáveis dos pinos de entrada e saída.	104
59	Funções de acesso aos pinos de entrada e saída.	104
60	Configuração dos pinos da FPGA referente ao controlador Ethernet.	104
61	Configuração dos pinos dos controladores UART_0, UART_1 e UART_2.	105
62	Configuração dos pinos do controlador SPI.	105
63	Análise dos componentes utilizados no projeto.	105
64	Interface principal de configuração.	106
65	Interface de definição do <i>hardware</i>	107
66	Exemplo de suporte oferecido pelo μ Clinux aos diversos fabricantes.	107
67	Opções de configuração do μ Clinux.	107
68	Bibliotecas de configuração dos periféricos.	108
69	Configuração dos aplicativos do μ Clinux.	108
70	Sub-blocos do componente BusyBox.	109
71	Componentes disponíveis no sub-bloco Coreutils.	109
72	Opção de seleção do processador e memória de armazenamento.	111

73	Interface usuário para programação do projeto na FPGA.	111
74	Definição da interface de comunicação entre o microcomputador e o kit DE2.	112
75	Interface usuário da licença do ambiente Quartus II.	112
76	Mensagem apresentada pelo Quartus II referente à licença.	113
77	Exemplo de transferência do arquivo sof.	113
78	Transferência da imagem do μ Clinux para o kit Nios II.	113
79	Sistema operacional μ Clinux sendo inicializado.	114
80	Sistema operacional μ Clinux em modo de comando.	115
81	Informações da CPU visualizadas através do ambiente μ Clinux.	115
82	Interrupções visualizadas através do ambiente μ Clinux.	115
83	Versão do sistema operacional μ Clinux.	116
84	Dispositivos do nó de rede.	116
85	Configuração da rede Ethernet.	117
86	Comandos para manipulação do servidor FTP.	117
87	Transferência de arquivo para o servidor FTP.	117
88	Arquivos de configuração do ambiente μ Clinux.	118
89	Diretório de armazenamento das páginas web no ambiente μ Clinux.	118
90	Página web armazenada no μ Clinux.	119
91	Interface principal do <i>software</i> NCAP.	120
92	Interface de descrição dos comitês do padrão IEEE 1451.	120
93	Fluxograma da leitura dos TEDS.	121
94	Interface de seleção da leitura dos TEDS.	121
95	Exemplo de leitura dos TEDS armazenadas em arquivo no NCAP.	122
96	Descrição dos TEDS referente ao módulo STIM1_RS232.	123
97	Campos para controle ou monitoramento dos transdutores conectados ao TIM.	123
98	Fluxograma do <i>software</i> de acesso aos transdutores inteligentes.	124

99	Forma de onda do comando enviado do NCAP para o STIM1_RS232.	124
100	Forma de onda do comando enviado do STIM1_RS232 para o NCAP.	124
101	Resposta do módulo STIM1_RS232 referente a temperatura.	125
102	Forma de onda do comando enviado do NCAP para o STIM1_RS232.	125
103	Forma de onda do comando enviado do STIM1_RS232 para o NCAP.	125
104	Resposta do módulo STIM1_RS232 referente ao motor de passo.	126
105	Forma de onda do comando enviado do NCAP para o WTIM1_ZigBee.	126
106	Forma de onda do comando enviado do WTIM1_ZigBee para o NCAP.	126
107	Resposta do módulo WTIM1_ZigBee referente ao controle do motor CC.	127
108	Foto do sistema desenvolvido em laboratório.	127
D.1	Sensor LM35 visualizado de cima.	171
D.2	Foto do motor de passo utilizado em laboratório.	172
E.1	Circuito STIM_1 desenvolvido em laboratório.	174
E.2	Circuito STIM_2 desenvolvido em laboratório.	175
E.3	Circuito WTIM_1 desenvolvido em laboratório.	176
E.4	Circuito WTIM_2 desenvolvido em laboratório.	177
F.1	Módulo XBee-Pro.	178
F.2	Módulo CON-USBee.	179
F.3	Módulo CON-USBee com XBee-Pro conectado ao microcomputador.	179
F.4	Interface usuário apresentada pelo ambiente X-CTU.	179
F.5	Teste de comunicação com módulo CON-USBee.	180
F.6	Interface usuário de configuração do módulo XBee-Pro.	180
F.7	Definição do nome do módulo XBee-Pro.	181
F.8	Configuração da velocidade de transmissão do módulo XBee-Pro.	181
F.9	Configuração do buffer de caracteres.	182

Lista de Tabelas

1	Formato padrão dos TEDS.	43
2	Ordem de transmissão dos dados.	44
3	Exemplo de bit mais significativo e menos significativo.	44
4	Estrutura de uma mensagem de comando	45
5	Classe de comando.	45
6	Comando de operação do transdutor.	46
7	Mensagem de comando de resposta.	46
8	Relação de frequência máxima em MHz entre os processadores Nios II. . .	75
9	Relação das MIPS de acordo com cada processador Nios II.	76
10	Característica de cada kit DE.	79
C.1	Meta-TEDS.	154
C.2	TransducerChannel TEDS - LM35.	157
C.3	TransducerChannel TEDS - Motor de passo 1.	161
C.4	User's Transducer Name TEDS.	162
C.5	PHY TEDS.	163
C.6	Meta-TEDS.	164
C.7	TransducerChannel TEDS - Ventilador 1.	165
C.8	User's Transducer Name TEDS.	166
C.9	Meta-TEDS.	166

C.10 User's Transducer Name TEDS.	167
C.11 PHY TEDS.	168
C.12 Meta-TEDS.	169
C.13 User's Transducer Name TEDS.	170
D.1 Passo completo.	172
D.2 Meio passo.	172

Lista de Abreviaturas e Siglas

A/D	<i>Analogic/Digital</i>
ABS	<i>Antilock Braking System</i>
ADC	<i>Aanalogic-to-Digital Converter</i>
AGC	<i>Apollo Guidance Computer</i>
ATM	<i>Asynchronous Transfer Mode</i>
BCPL	<i>Basic Combined Programming Language</i>
bps	Bits por segundo
CAN	<i>Controller Area Network</i>
CC	Corrente Contínua
CGI	<i>Common Gateway Interface</i>
CI	Circuito Integrado
CiA DS 404	<i>CAN in Automation – Device Specification 404</i>
CMOS	<i>Complementary Metal-Oxide Semiconductor</i>
CPLD	<i>Complex Programmable Logic Device</i>
CPU	<i>Central Processing Unit</i>
CRC	<i>Cyclic Redundancy Code</i>
CSMA/CD	<i>Carrier Sense Multiple Access with Collision Detection</i>
D/A	<i>Digital/Analogic</i>
DDR DRAM	<i>Double Data Rate Dynamic Random-Access Memory</i>
DE1	<i>Development Education 1</i>

DE2	<i>Development and Education 2</i>
DMA	<i>Direct Memory Access</i>
DMC	<i>Distributed Measurement and Control</i>
DSSS	<i>Direct-Sequence Spread Spectrum</i>
eCos	<i>Embedded Configurable Operating System</i>
EDA	<i>Environmental Development Association</i>
EIA	<i>Electronic Industries Alliance</i>
EPCS	<i>Embedded Programming the Serial Configuration Chip</i>
EPOS	<i>Embedded Parallel Operating System</i>
FDDI	<i>Fiber Distributed Data Interface</i>
FFDs	<i>Full Function Devices</i>
FHSS	<i>Frequency-Hopping Spread Spectrum</i>
FPGA	<i>Field-Programmable Gate Array</i>
FTP	<i>File Transfer Protocol</i>
GHz	<i>Giga-Hertz</i>
GPS	<i>Global Positioning System</i>
HCPLD	<i>High Complex Programmable Devices</i>
HTML	<i>HyperText Markup Language</i>
HTTP	<i>HyperText Transfer Protocol</i>
I/O	<i>Input/Output – Entrada/Saída</i>
I2C	<i>Inter-Integrated Circuit</i>
ID	<i>Identification</i>
IEEE	<i>Institute Of Electrical And Electronics Engineers - Instituto de Engenheiros Eletricistas e Eletrônicos</i>
IP	<i>Internet Protocol</i>
IR	<i>Infrared</i>
IrDa	<i>Infrared Data Association</i>
IRQ	<i>Interrupt Request</i>
ISM	<i>Industrial, Scientific and Medica</i>

ISO/IEC	Organização Internacional para Padronização / Comissão Eletrotécnica Internacional
JTAG	<i>Joint Test Action Group</i>
Kbps	<i>Kilobits Per Second</i>
LAN	<i>Local Area Network</i>
LCD	<i>Liquid Crystal Display</i>
LLC	<i>Logical Link Control</i>
mA	<i>Miliampère</i>
MAC	<i>Medium Access Control</i>
MB	Mega Bytes
MCI	<i>Module Communications Interface</i>
MCU	<i>Micro Controller Unit</i>
MHz	Mega-Hertz
MIMOSA	<i>Machinery Information Management Open Systems Alliance</i>
MIPS	<i>Millions of Instructions Per Second</i>
MISO	<i>Master In / Slave Out Pin</i>
MIT	<i>Massachusetts Institute of Technology</i>
MMI	<i>Mixed-Mode Interface</i>
MMU	<i>Memory Management Unit</i>
MOSI	<i>Master Out / Slave In Pin</i>
Mpbs	<i>Megabit Per Second</i>
NCAP	<i>Network Capable Application Processor</i>
NIST	<i>National Institute of Standards and Technology</i>
NTP	<i>Network Time Protocol</i>
OEMs	<i>Original Equipment Manufacturer</i>
OGC-SWE	<i>Open Geospatial Consortium Sensor Web Enablement</i>
OSA-CBM	<i>Open System Architecture for Condition Based</i>

Maintenance

OSI	<i>Open Systems Interconnection</i>
OVI	<i>Open Verilog International</i>
PAN	<i>Personal Area Network</i>
PDA _s	<i>Personal Assistant Digital</i>
PIC	<i>Programmable Interface Controller</i>
PID	<i>Process Identification</i>
PPP	<i>Point-to-Point Protocol</i>
PSRS	<i>Parallel Sorting by Regular Sampling</i>
PTP	<i>Precision Time Protocol</i>
PWM	<i>Pulse-Width Modulation</i>
RAM	<i>Random Access Memory</i>
RFD _s	<i>Reduced Function Devices</i>
RFID	<i>Radio-Frequency Identification</i>
RISC	<i>Reduced Instructions Computer</i>
ROM	<i>Read Only Memory</i>
rpm	Rotação por minuto
RS-232	<i>Recommended Standard-232</i>
RTL	<i>Register Transfer Level</i>
SCADA	<i>Supervisory Control and Data Acquisition</i>
SCK	<i>Serial Clock Pin</i>
SD Card	<i>Secure Digital Card</i>
SDRAM	<i>Synchronous Dynamic Random Access Memory</i>
SNAP	<i>Scalable Node Address Protocol</i>
SOA	<i>Service-Oriented Architecture</i>
SoC	<i>Security Operations Center</i>
SoPC	<i>System on a Programmable Chip</i>
SPI	<i>Serial Peripheral. Interface</i>
SPIV3	<i>Serial Peripheral Interface Bus Version 3</i>

SRAM	<i>Static Random Access Memory</i>
SS	<i>Slave Select Pin</i>
STIM	<i>Smart Transducer Interface Module</i>
T/L/V	<i>Type/Length/Value</i>
TBC	<i>Transducer Bus Controller</i>
TBIM	<i>Transducer Bus Interface Module</i>
TCP/IP	<i>Transfer Control Protocol / Internet Protocol</i>
TEDS	<i>Transducer Electronic Data Sheet</i>
Telnet	<i>Telecommunications Network</i>
TII	<i>Transducer Independent Interface</i>
TIM	<i>Transducer Interface Module</i>
UART	<i>Universal Asynchronous Receiver/Transmitter</i>
UART-STIM	<i>Universal Asynchronous Receiver/Transmitter-Smart Tranducer Interface Module</i>
UFM	<i>User Flash Memory</i>
UML	<i>Unified Modeling Language</i>
USB	<i>Universal Serial Bus</i>
UUID	<i>Universal Unique Identifier</i>
Verilog HDL	<i>Verilog Hardware Description Language</i>
Verilog LRM	<i>Verilog Language Reference Manual</i>
VGA	<i>Video Graphics Array</i>
VHDL	<i>VHSIC Hardware Description Language</i>
Wi-Fi	<i>Wireless Fidelity</i>
WLAN	<i>Wireless Local Area Network</i>
WTIM	<i>Wireless Transducer Interface Module</i>

Sumário

1	Introdução	26
1.1	Redes Industriais	28
1.2	Objetivos e Estratégias de Desenvolvimento	29
1.3	Justificativa	30
1.4	Trabalhos Relacionados à Área	31
1.5	Organização do Texto	40
2	O Padrão IEEE 1451	41
2.1	Introdução	41
2.2	O Padrão IEEE 1451.0 - 2007	41
2.2.1	Formato padrão dos TEDS	43
2.2.2	Estrutura das Mensagens	44
2.2.3	Comandos	46
2.3	O Padrão IEEE 1451.1	47
2.3.1	Processador de Aplicação com Capacidade de Operar em Rede (NCAP) - IEEE 1451.1	48
2.3.1.1	Modelo de Informação	49
2.3.1.2	Modelo de Dados	50
2.3.1.3	Modelos de Comunicação em Rede	51
2.4	O Padrão IEEE P1451.2	51

2.5	O Padrão IEEE 1451.3	52
2.6	O padrão IEEE 1451.4	53
2.7	O padrão IEEE 1451.5	54
2.8	O padrão IEEE P1451.6	55
2.9	O padrão IEEE 1451.7	56
2.10	Considerações Finais Sobre o Capítulo 2	56
3	Interfaces de Comunicação Utilizadas no Nó de Rede IEEE 1451	58
3.1	Introdução	58
3.2	Comunicação Serial	58
3.2.1	<i>Bit Rate e Baud Rate</i>	60
3.2.2	O Padrão RS-232	60
3.3	<i>Serial Peripheral Interface Bus - SPI</i>	61
3.4	ZigBee - Padrão IEEE 802.15.4	62
3.4.1	Relação entre o ZigBee e o IEEE 802.15.4	63
3.4.2	Tipos de dispositivos	64
3.4.3	Topologia de Rede ZigBee	64
3.5	O Padrão IEEE 802.3 e a Ethernet	66
3.5.1	Estrutura do quadro Ethernet	67
3.5.2	<i>CSMA/CD: Carrier Sense Multiple Access with Collision Detection</i>	68
3.6	Considerações Finais Sobre o Capítulo 3	69
4	Ferramentas Utilizadas no Desenvolvimento do Nó de Rede NCAP e dos TIMs	70
4.1	Introdução	70
4.2	Sistema Operacional Embarcado μ Clinux	70
4.3	Ambiente SoPC Builder	72
4.4	Verilog HDL	73

4.5	Processador Nios II	74
4.6	Dispositivo Lógico Programável e a plataforma de Desenvolvimento DE2	77
4.7	O Servidor Boa	79
4.8	<i>Commom Gateway Interface</i> - CGI	80
4.9	BusyBox	80
4.10	Os Microcontroladores	81
4.11	Linguagem C	81
4.12	Considerações Finais Sobre o Capítulo 4	83
5	Descrição dos TIMs Desenvolvidos para Testes com o Nó de Rede Embarcado NCAP	84
5.1	Introdução	84
5.2	Descrição Geral de Desenvolvimento dos TIMs - IEEE 1451	84
5.3	<i>Smart Transducer Interface Module</i> - STIM	87
5.4	<i>Wireless Transducer Interface Module</i> - WTIM	89
5.5	Testes realizados com os STIMs e WTIMs	91
5.6	Considerações Finais Sobre o Capítulo 5	94
6	Desenvolvimento do Nó de Rede IEEE 1451.1	95
6.1	Introdução	95
6.2	O Nó IEEE 1451.1 Embarcado	95
6.3	Descrição do <i>Hardware</i>	97
6.3.1	Definição e Configuração do <i>Hardware</i>	98
6.4	Sistema Operacional Embarcado	106
6.4.1	Interface de Configuração (make menuconfig)	106
6.4.2	Transferência do <i>Hardware</i> e da Imagem para Kit de Desenvolvimento DE 2	110
6.5	Ambiente μ Clinux	114
6.6	<i>Software</i> NCAP	118

7 Conclusões Gerais	128
7.1 Conclusões	128
7.2 Contribuições	130
7.3 Sugestões de Trabalhos Futuros	130
Referências	132
A μClinux-dist	137
A.1 Instalação do μ Clinux-dist	137
A.2 Configuração da imagem do μ Clinux	139
A.2.1 Configuração das Interfaces UART no Ambiente uClinux-dist	140
A.2.2 Configuração da Interface SPI no Ambiente uClinux-dist	141
A.2.3 Configuração do Servidor Web, FTP e Acesso Remoto Telnet	142
A.2.4 Compilação de arquivos em linguagem C para ambiente μ Clinux	143
B Comandos de Leitura e Controle	144
B.1 Comando para Monitoramento e Controle do Módulo STIM_1	144
B.2 Comando para Monitoramento e Controle do Módulo STIM_2	146
B.3 Comando para Monitoramento e Controle do Módulo WTIM_1	148
B.4 Comando para Monitoramento e Controle do Módulo WTIM_2	149
B.5 Comandos de Requisição e Resposta dos TEDS	150
C Transducer Eletronic Data Sheet	153
C.1 STIM_1	153
C.1.1 Meta-TEDS	153
C.1.2 TransducerChanel TEDS	154
C.1.2.1 Sensor LM35	155
C.1.2.2 Motor de passo 1	158
C.1.2.3 Motor de passo 2	160

C.1.3	User's Transducer Name TEDS	160
C.1.4	PHY TEDS	162
C.2	STIM_2	163
C.2.1	Meta-TEDS	163
C.2.2	TransducerChanel TEDS	163
C.2.2.1	Ventilador 1	164
C.2.2.2	Ventilador 2	164
C.2.2.3	Ventilador 3	164
C.2.3	User's Transducer Name TEDS	164
C.2.4	PHY TEDS	165
C.3	WTIM_1	165
C.3.1	Meta-TEDS	165
C.3.2	TransducerChanel TEDS	166
C.3.2.1	Sensor LM35	166
C.3.3	User's Transducer Name TEDS	166
C.3.4	PHY TEDS	167
C.4	WTIM_2	169
C.4.1	Meta-TEDS	169
C.4.2	TransducerChanel TEDS	169
C.4.2.1	Sensor LM35	169
C.4.2.2	Motor CC	169
C.4.3	User's Transducer Name TEDS	170
C.4.4	PHY TEDS	170
D	Descrição dos Transdutores	171
D.1	LM35	171
D.2	Motor de Passo	171

D.3 Ventiladores	173
D.4 Motor CC	173
E Cricuitos	174
E.1 Circuito STIM_1	174
E.2 Circuito STIM_2	175
E.3 Circuito WTIM_1	176
E.4 Circuito WTIM_2	177
F Configurações do Módulo XBee-Pro	178

Capítulo 1

Introdução

Num microcomputador, para realizar o interfaceamento do usuário com o *hardware*, utiliza-se um *software* denominado sistema operacional, que é responsável pelo gerenciamento dos aplicativos no acesso ao *hardware*. Tais sistemas operacionais são desenvolvidos e aplicados em diferentes arquiteturas de microcomputadores, sem a necessidade de se preocupar com as características de *hardware*.

Diferentes dos microcomputadores, os dispositivos embarcados possuem uma arquitetura restrita, tal como de memória, interface com usuário (*displays*), I/O (*Input/Output* - entrada/saída) e ao processamento. Os *softwares* desenvolvidos para a implementação nesses dispositivos é dedicado e encapsulados no dispositivo realizando um conjunto de tarefas pré-definidas, geralmente com requisitos específicos. Os sistemas embarcados, muitas vezes, são utilizados para realizar o interfaceamento de transdutores (sensor/atuador) e microcomputadores, ou simplesmente para a manipulação de um dispositivo ou a realização da aquisição de dados. Como exemplo de aplicações utilizando sistemas embarcados, têm-se:

- Medicina:
 - Máquinas de hemodiálise, monitores cardíacos e máquinas de Raio-X.
- Aparelhos domésticos:
 - Televisores analógicos e digitais, PDAs (*Personal Digital Assistant*), celulares, VOIP (*Voice Over Internet Protocol*), jogos, brinquedos, câmeras e GPS (*Global Position System*).
- Periféricos de computadores:

- Roteadores, *switches*, *gateways*, impressoras, *scanners*.
- Automotivo:
 - Ignição, sistema de freio ABS (*Antiblockier-Bremssystem*).
- Instrumento Musical:
 - Sintetizadores, pianos.
- Aeroespacial:
 - Sistema de navegação.
- Agricultura:
 - Tratores e implementos agrícolas.

O gerenciamento e o acesso ao *hardware* dos dispositivos embarcados podem ser realizados utilizando um sistema operacional embarcado, que é empregado utilizando microcontrolador, microprocessadores ou FPGA (*Field Programmable Gate Array*), tecnologia denominada SoC (*System-on-Chip*). Os sistemas operacionais embarcados são responsáveis por realizar o gerenciamento de I/O, memórias, processadores, entre outros, além de oferecerem uma máquina virtual que tem como objetivo mascarar as rotinas de acesso ao *hardware*. Como exemplos de sistemas operacionais embarcados, têm-se: μ Clinux (JUNQUEIRA, 2007), eCos (*Embedded Configurable Operating System*) (ECOS, 2002), Windows XP embedded (STADZISZ, 2003), e C Executive (SYSTEMS, 1998).

Atualmente, os sensores e atuadores são utilizados em conjunto com os sistemas embarcados, denominando-os como transdutores inteligentes, e são utilizados no controle ou/e na aquisição dos dados em sistemas DMC (*Distributed Measurement and Control*). O desenvolvimento de sistemas DMC demanda um grande esforço por parte de engenheiros e projetistas da área de instrumentação, sendo que alguns desses sistemas são desenvolvidos para uma determinada aplicação, utilizando ferramentas de alto custo e *softwares* proprietários. Uma característica de sistema que surgiu no início da década de 80, com a evolução das comunicações digitais na área de automação industrial. Tais sistemas de interconexão possibilitaram a comunicação digital bidirecional trabalhando com um determinado protocolo, permitindo troca de informação entre os módulos de transdutores inteligentes. Este fato facilitou o avanço na automação industrial, porém, surgiram inúmeros problemas, como, por exemplo, o interfaceamento dos transdutores e as diferentes tecnologias de rede (SONG et al., 2011).

Ao analisar os esforços no desenvolvimento de sistemas DMC, no início da década de 90, foram aprovadas as primeiras diretrizes para o padrão IEEE (*Institute of Electrical and Electronic Engineers*) 1451, um padrão de interfaceamento para transdutores inteligentes. As diretrizes foram desenvolvidas pelo IEEE em parceria com o NIST (*National Institute of Standards and Technology*), para a conexão de transdutores inteligentes em rede. Após essa parceria entre a NIST e a IEEE, diversos Workshops e congressos foram realizados definindo, assim, conceitos no padrão IEEE 1451. Hoje, o padrão IEEE 1451 possui diversos conceitos de como conectar transdutores em rede e é dividido nos seguintes comitês: IEEE 1451.0, IEEE 1451.1, IEEE P1451.2, IEEE 1451.3, IEEE 1451.4, IEEE 1451.5, IEEE P1451.6 e IEEE 1451.7 (SONG et al., 2011).

O estudo da tecnologia dos dispositivos embarcados e o gerenciamento de *hardware* através dos sistemas operacionais embarcados norteou o desenvolvimento do NCAP (*Network Capable Application Processor*), com diferentes características e funcionalidades. Tal NCAP foi capaz de obter dados de uma rede externa e interagir com módulos de transdutores inteligentes utilizando diferentes interfaces de comunicação de acordo com os comitês IEEE P1451.2 e IEEE 1451.5. O *hardware* e o sistema operacional embarcado implementado para o desenvolvimento do nó de rede foram definidos de forma dinâmica baseando-se na norma definida pelo padrão IEEE 1451 e nas características das interfaces de comunicação. Na Seção 1.1 apresenta-se um breve histórico das redes industriais e sua evolução.

1.1 Redes Industriais

A competitividade e a busca constante de minimização de custos nas fábricas demonstram o constante e acelerado desenvolvimento na área de automação industrial. Durante a década de 40, a instrumentação dos processos industriais era realizada utilizando ar-comprimido trabalhando com faixas de valores analógicos entre 3-15 psi¹, para realizar o controle e o monitoramento dos equipamentos (LIMA, 2004).

Com a evolução dos sistemas eletrônicos, a década de 60 ficou marcada pelo surgimento de sinais analógicos elétricos, trabalhando com uma faixa de valores de 4-20 mA (*miliampère*). Na década de 70, os microcomputadores foram introduzidos nas fábricas de forma centralizada, realizando a aquisição de dados e controlando dispositivos. A partir desta década, também começaram a surgir os primeiros *softwares* SCADA (*Supervisory Control and Data Acquisition*), possuindo diversas funcionalidades e podendo ser aplicados

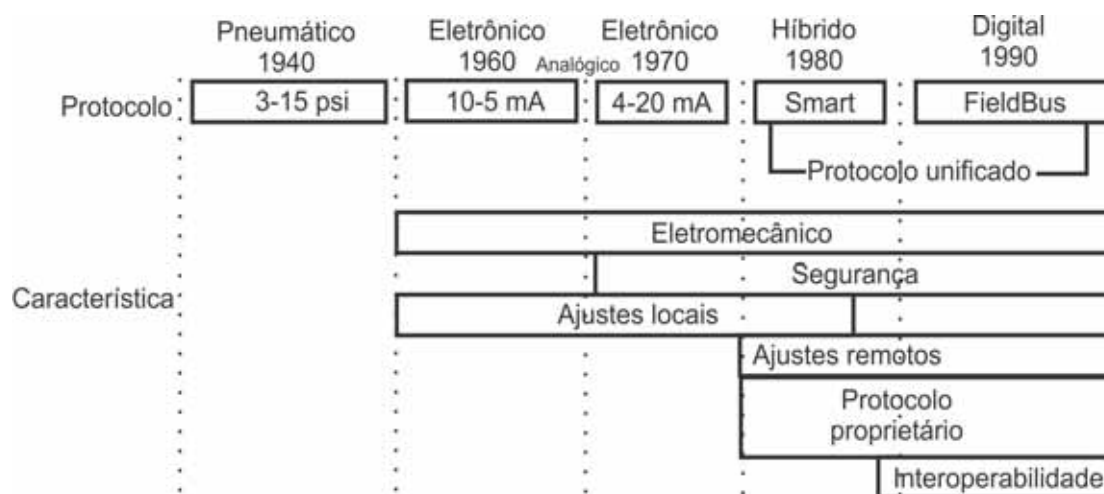
¹psi - libra por polegada quadrada, unidade de medida americana.

em diferentes sistemas operacionais (LIMA, 2004).

No decorrer dos anos, com o avanço da tecnologia de integração de componentes eletrônicos, foram surgindo microprocessadores cada vez mais poderosos e com menor custo financeiro. Com isso, cada transdutor passou a ser implementado com processador dedicado, dando origem a um novo conceito de transdutores, denominado de transdutores inteligentes. Implementar diversos tipos de transdutores em uma rede buscando melhorar o desempenho, com maior quantidade e qualidade dos dados, tornou-se possível, dando origem ao barramento de campo (LIMA, 2004).

Com o avanço da tecnologia do barramento de campo, muitas empresas passaram a desenvolver sua própria rede de transdutores baseando-se apenas nos seus interesses, acarretando, assim, outros problemas, como o interfaceamento dos transdutores e as diferentes tecnologias de rede (LIMA, 2004). Na Figura 1, é ilustrada a evolução das redes de transdutores demonstrando a década, o protocolo utilizado e a característica do sistema.

Figura 1 – Evolução das redes de transdutores inteligentes.



Fonte: Lima (2004)

1.2 Objetivos e Estratégias de Desenvolvimento

O objetivo principal do presente trabalho é o desenvolvimento de um nó de rede capaz de obter dados de uma rede externa, independente de tecnologia ou protocolo de comunicação, processar esses dados e enviar para uma de suas interfaces de comunicação com TIM (*Transducer Interface Module*), para realizar a leitura ou o controle dos transdutores inteligentes, de acordo com o padrão IEEE 1451.

No desenvolvimento do nó de rede, foi utilizado o ambiente SoPC Builder (*System on a Programmable Chip*), permitindo ao projetista definir a arquitetura do nó, de acordo com as características do sistema, utilizando componentes pré-definidos. Para a implementação e testes do sistema foi utilizado o kit DE2 que possui interfaces de comunicação para outros dispositivos, pinos de I/O, dispositivo lógico programável e memórias.

Além das características de *hardware* dinâmico, o nó de rede foi desenvolvido utilizando o sistema operacional μ Clinux, que possui código aberto, gratuito e dinâmico, tendo sido gerado baseado nas características de *hardware* e de acordo com a necessidade da aplicação em que será utilizado. O sistema operacional possui características semelhantes a um sistema operacional Linux voltado para microcomputadores, porém, todos os seus comandos básicos devem ser definidos na compilação do *kernel* do sistema. Tais características de *hardware* e do sistema operacional embarcado, oferecem suporte ao desenvolvimento de um nó de rede baseado na norma IEEE 1451 com diferentes interfaces de forma dinâmica e que possa ser aplicada em diversos ambientes em condições adversas.

Para os testes do nó de rede, foram desenvolvidos módulos TIMs, utilizando interface com fio baseado no padrão IEEE P1451.2 e sem fio baseado no padrão IEEE 1451.5. O objetivo foi verificar a integridade dos dados e do nó de rede utilizando diferentes interfaces de comunicação. Outro aspecto avaliado foi a capacidade de *plug and play*, em que, ao conectar o TIM no nó de rede NCAP, as informações dos TEDS (*Transducer Electronic Data Sheet*) são transferidas para o NCAP realizando o reconhecimento dos transdutores.

Para o controle e monitoramento de transdutores inteligentes, o usuário pode acessar o sistema através de páginas web, pois, através do servidor Boa, foi possível disponibilizar páginas web no nó de rede embarcado. Para realizar as configurações do sistema, o administrador da rede pode acessar o nó de rede através do acesso remoto, utilizando o protocolo Telnet (*Telecommunications Network*) disponibilizando acesso independente do ponto da rede.

1.3 Justificativa

Um sistema de instrumentação e controle distribuído, utilizando o padrão de comunicação (IEEE 1451) e a filosofia de sistemas abertos, tem características como flexibilidade, interoperabilidade, *plug and play* e reutilização de códigos. Utilizando o padrão IEEE 1451, as indústrias poderão obter inúmeros benefícios como a redução de custos de sistemas de instrumentação, maior flexibilidade e facilidade na implementação de novos módulos. Além das indústrias, o nó de rede pode ser aplicado em diversas outras

áreas como a biomédica, telemedicina e automação residencial.

As diferentes formas de conexões entre o NCAP e TIMs para os sistemas de instrumentação e controle distribuído norteiam o desenvolvimento de um nó NCAP com capacidade de interligar com diferentes blocos TIMs através de diferentes interfaces de comunicação. Para o nó de rede NCAP, a descrição de um *hardware* dinâmico e a instalação de um sistema operacional μ Clinux minimizam os custos de desenvolvimento e facilitam o gerenciamento dos processos, além de realizar o controle dos dados enviados através de uma rede externa.

A criação de diferentes TIMs, para realizar a comunicação com um único NCAP através de diferentes interfaces (padrão IEEE 1451), facilita a implantação do sistema para realização de diferentes funcionalidades, por exemplo, o controle de um motor de passo, utilizando rede cabeada, padrão IEEE P1451.2 e o monitoramento da temperatura de uma estufa utilizando rede sem fio padrão IEEE 1451.5. Outro fator que viabiliza o sistema é a característica *plug and play* dos TIMs, sendo que, para cada ambiente, pode ser adicionado um novo módulo (IEEE P1451.2 e IEEE 1451.5) no NCAP embarcado, independente da interface de comunicação, para que possa realizar o controle ou o monitoramento dos transdutores, de acordo com o padrão IEEE 1451.

1.4 Trabalhos Relacionados à Área

Na área dos sistemas operacionais embarcados, um trabalho que teve contribuição relevante foi o de Bueno, Brasil e Marques (2007). Trata-se da implementação de um sistema operacional embarcado eCos no kit Nios II Altera, para avaliação do desempenho do processador Nios II, operando com processamento paralelo. Para analisar o desempenho do processador Nios II, foram utilizados três algoritmos, sendo dois de ordenação, o PSRS (*Parallel Sorting by Regular Sampling*) e Quicksort e um algoritmo de multiplicação de matrizes. Os experimentos que foram realizados demonstraram as características em relação à eficiência e *speedup*, sendo que foram obtidos os melhores resultados no algoritmo de ordenação PRSR. O trabalho apresentou também a facilidade de se implementar processamento paralelo em ambiente embarcado, como por exemplo, em sistema de robótica embarcada, cujos algoritmos necessitam de uma grande quantidade de processamento.

Junqueira (2007) apresentou a implementação do sistema operacional μ Clinux, utilizando o kit de desenvolvimento Nios II como proposta de desenvolvimento de um PDA (*Personal Digital Assistant*). Como primeiro passo, foi definida a arquitetura do

hardware no ambiente SoPC Builder. Após a definição da arquitetura, realizou-se a compilação do kernel do μ Clinux, com a arquitetura de descrição de *hardware* gerada pelo SoPC Builder, através de um arquivo que contém a descrição do *hardware*. Ao gerar a imagem do μ Clinux, os arquivos foram transferidos para o kit de desenvolvimento Nios II através do modo de comando disponibilizado pelo *software* Nios IDE (*Integrated Drive Electronics*). Ao finalizar a transferência do *hardware* e da imagem para a placa, foram realizados testes de visualização do ambiente gráfico através da saída VGA (*Video Graphics Adapter*) e de um monitor de microcomputador, conexão com a rede Ethernet, utilizando o microcontrolador DM9000A, apresentação do servidor web e transferência de arquivo através do protocolo FTP (*File Transfer Protocol*).

No trabalho de Lutz e Faerber (2009), foi apresentada a arquitetura do dispositivo Nios II, suas características, como montar um processador Nios II com periféricos, instalação das ferramentas de compilação do sistema operacional embarcado μ Clinux. No trabalho, apresentaram-se as interfaces UART (*Universal Asynchronous Receiver/Transmitter*) e Ethernet. Os autores também descreveram a implementação do compilador do sistema operacional μ Clinux em ambiente Linux, ferramentas como *Toolchain*, bibliotecas necessárias para a compilação e escolha dos comandos e *software* para o ambiente de trabalho. Descreveram testes desenvolvidos no kit de desenvolvimento Nios II com sistema operacional μ Clinux, testes de acesso aos periféricos, serviços de rede e configurações dos periféricos.

Wiedenhof, Hoeller e Frohlich (2007) apresentaram o desenvolvimento de um gerenciador de energia, utilizando o sistema operacional embarcado EPOS (*Embedded Parallel Operating System*). O trabalho também abordou as características do sistema EPOS e a importância do gerenciamento de energia em sistemas embarcados. Apesar de utilizar o sistema EPOS, o gerente é um *software* à parte para controle de energia. Os testes foram realizados utilizando um microcontrolador ATmega16 da Atmel e vários testes foram realizados como, equipamento desligado, aplicação sem gerência de energia, com gerência de energia dirigida pela aplicação, disponibilizada pela infraestrutura de gerência EPOS e com o gerente de energia desenvolvida no trabalho. Os testes foram realizados utilizando aplicações determinísticas e não determinísticas. Com os resultados dos testes, o trabalho resultou em um melhor resultado com aplicações não determinísticas.

No trabalho de Thorne (2007) implementou-se o sistema operacional embarcado μ Clinux em uma placa com uma FPGA modelo 2c35 (Altera). O trabalho teve como objetivo desenvolver um *driver* para um teclado de 16 teclas para μ Clinux e realizar a análise de performance que o *soft-core* traz para os sistemas embarcados. O trabalho

apresentou as camadas de *software* e *hardware*, que foram implementadas, os comandos de compilação do kernel do μ Clinux e como adicionar módulos no sistema operacional. Para testes do sistema, ao desenvolver o *driver* para placa 2c35, foi instalado o sistema operacional μ Clinux em uma outra placa com uma FPGA modelo 1c12 e foi realizada a instalação do *driver* e reconhecido o dispositivo pelo sistema operacional.

No trabalho de Sigla e Morris (2008), foram apresentados testes de comunicação entre dois NCAPs distintos, de acordo com as normas do padrão IEEE 1451. Os autores fizeram uma breve descrição do padrão IEEE 1451.1, padrão IEEE P1451.2 e apresentaram uma proposta de implementação entre dois blocos NCAPs conectados entre si em rede. No trabalho, foi apresentado o mecanismo utilizado para realizar a comunicação entre os dois NCAPs, na qual a mensagem é codificada e transmitida para o nó de destino, que decodifica essa informação localmente. Os autores fizeram também uma abordagem da quantidade de pacotes que são enviados por todas as operações na rede de transdutores e ressaltaram que os pacotes que são enviados entre nós devem conter um cabeçalho único e padronizado. O trabalho fez referência a três principais tipos de cabeçalho: pacote público, cliente/servidor e servidor/cliente. Os testes de comunicação entre os dois NCAPs foram apresentados através de relatórios, informando o que houve em cada pacote de comunicação como, por exemplo, identificação do endereço IP (*Internet Protocol*) e endereço MAC (*Medium Access Control*).

Wobschall e Mupparaju (2008) desenvolveram em seu trabalho o protocolo SNAP (*Scaleable Node Address Protocol*) para transferência de dados entre o módulo NCAP e TIM, utilizando a interface RFID (*Radio-Frequency Identification*) de acordo com o padrão IEEE 1451. Para realização do trabalho, utilizaram um módulo PIC (*Programmable Interface Controller*) 12F675F, trabalhando com uma frequência de 433 MHz, um módulo Chipcon/TI CC1100 operando em 433 MHz e com interface serial RS-232 (*Recommended Standard-232*). No trabalho apresentado, os autores abordaram sobre as características do protocolo SNAP, apresentaram o formato dos campos e as suas especificações. Os autores também relataram que, se dois sensores transmitem simultaneamente, os dados são perdidos. Se houver problemas com ruídos resultando em erro no CRC, haverá perda de dados. Para testes do sistema, foram realizadas aquisições da temperatura em um intervalo de 5 minutos com uma bateria de 3 V, o tempo de permanência do sistema foi de 2 meses. Com os testes realizados, os autores puderam concluir que o consumo energia é 10 vezes menor, utilizando o protocolo SNAP.

Wobschall (2008) apresentou em seu trabalho um NCAP desenvolvido em um microcomputador, realizando a comunicação entre o NCAP e o TIM através da interface

serial. O módulo de comunicação entre o NCAP e o microcomputador foi realizado utilizando a rede Ethernet e visualizando os dados através da Internet, usando um navegador web. Para esse modelo de rede de transdutores inteligentes, utilizou-se um sensor de temperatura e um fotodiodo (sensores) e um relê como atuador. Em outro exemplo disponibilizado pelo autor, implementou-se o padrão IEEE 1451.5, para realizar a comunicação entre o NCAP e WTIM (*Wireless Transducer Interface Module*), utilizando os mesmos módulos e os mesmos protocolos. Além dos testes realizados, o autor fez uma abordagem sobre as TEDS e sobre os *softwares* desenvolvidos no projeto.

Song et al. (2011) desenvolveram um NCAP em um *notebook* capaz de realizar a comunicação com dois WTIMs, utilizando o padrão IEEE 1451.5-802.11 e o padrão IEEE 1451.0 para especificação dos TEDS e protocolos. No trabalho, os autores apresentaram testes realizados entre os dois blocos NCAP e WTIM, como reconhecimento dos módulos na rede, definindo um ID (*Identification*) e endereço IP para cada WTIM e o reconhecimento de todos os transdutores conectados nos módulos. Ao demonstrar os transdutores ativos, foram gerados gráficos das leituras dos sensores e apresentados os gráficos dos dados colhidos. Na conclusão os autores abordam as limitações da placa utilizada para o desenvolvimento dos WTIMs como o tamanho do *buffer* disponível para o socket.

Viegas, Pereira e Girao (2008) descreveram o padrão IEEE 1451 e suas principais características, como o modelo de informação no qual é composto de uma hierarquia de classes dividido em três categorias principais: *Block*, *Component* e *Service*. No trabalho, foi descrito o modelo de dados, sendo: *Boolean*, *Octet*, *Integer*, *Float*, *String* e vetor de tipo de dados prévio, além da forma de comunicação em rede, sendo: um para um e um para muitos. Ao descrever as principais características do padrão IEEE 1451.1, os autores descreveram os principais blocos dentro de cada classe (*Block*, *Component* e *Service*) e para testes do sistema, foi feito um *software* de controle de nível de água em um tanque. Os autores também descreveram que, apesar das vantagens do padrão IEEE 1451.1, ainda não existem NCAPs comerciais disponíveis em mercado.

Em Ramos (2008), a autora apresenta o desenvolvimento de um NCAP com duas interfaces USB (*Universal Serial Bus*) implementado em um microcomputador. Como primeiro passo do trabalho, a autora realiza uma breve descrição do padrão IEEE 1451, cada comitê e quando as normas foram aprovadas. A segunda parte do trabalho apresentou o desenvolvimento do sistema em que cada TIM foi implementado usando o microcontrolador Freescale (HC9S12DT256) e uma memória (AT24C64A). Para a realização da comunicação entre o NCAP e o TIM, foi necessário a instalação dos *drivers*

USB para o nó de interface do FTDI (FT232BL) e o *software* LabView. O protocolo utilizado para a comunicação entre o NCAP e o TIM foi PTP (*Precision Time Protocol*) descrito pelo padrão IEEE 1588 no qual a autora descreve as características das mensagens e o tamanho. Na conclusão é apresentada as vantagens do uso da interface USB na comunicação entre o NCAP e o TIM e a possibilidade de desenvolvimento de interfaces *multi-point* no mesmo NCAP.

No trabalho de Song e Lee (2010) apresentaram a comunicação entre o NCAP e o WTIM, utilizando o padrão IEEE 1451.5, utilizando a interface sem fio baseada no padrão IEEE 802.11. Os autores descreveram as mensagens, comandos padrões de controle do TIM, comandos comuns, como por exemplo: leitura dos TEDS, escrita dos TEDS, requisição dos comandos dos TEDS etc., e comandos de resposta para o NCAP, todos representados por diagrama de classe. Como caso de estudo, os autores apresentaram o reconhecimento dos módulos WTIMs e apresentaram cada passo para reconhecimento dos PHY TEDS em uma interface gráfica feita em Java, validando o reconhecimento dos módulos.

Eccles (2008) apresentou um breve resumo de como são realizados os testes em aeronaves e como são obtidas essas medidas. Como parte da implementação do sistema para teste da estrutura e da reação destas estruturas para estas cargas, foram implementados 8000 sensores *strain gauge*. A princípio eram utilizados cabos do sensor da aeronave até as sala de testes, realizando, assim, o condicionamento de sinal, armazenamento dos dados e análises. Para sanar o problema de instalação dos cabos em aeronaves, foram utilizados pequenos multiplexadores, em que os sensores são conectados nesses multiplexadores em que foi realizado o condicionamento do sinal e a digitalização. O autor ressalta também que o problema dos cabos geralmente é mais uma função de números de horas de trabalho requeridas no laboratório e instalação do que uma função de materiais, então, o custo de cabos curtos não é significativamente menor do que de um cabo longo. Entretanto, o custo da instalação de cabos curtos é menor do que em cabos longos. O autor descreve uma solução para resolver a problemática dos cabos, utilizando redes de sensores inteligentes, porém, outra questão é levantada: o caso de que as redes de comunicação não são determinísticas. Como resposta em relação ao tempo de resposta da rede, o autor sugere dois tipos de protocolo: NTP (*Network Time Protocol*) e PTP. Para o desenvolvimento da rede, o autor descreve como uma solução a implementação do padrão IEEE 1451, porém, apresenta também os aspectos que impedem a implementação da norma na indústria de veículos e aeroespacial e a difícil aceitação da indústria devido a custos e às grandes mudanças que devem ser realizadas. Como conclusão, o autor descreve que alguns testes e avaliações em aeroplanos utilizando redes de transdutores inteligentes já

estão sendo realizado, porém, são desenvolvido utilizando padrões proprietários. O autor também descreve que o uso do padrão IEEE 1451 pode sanar os problemas utilizando PTP padrão IEEE 1588 para realização de testes utilizando redes de transdutores inteligentes em aeroplanos.

Tu (2009) apresentou um NCAP em linguagem C/C++ desenvolvido no ambiente C/C++ Builder 6.0. O TIM foi desenvolvido baseado no padrão IEEE P1451.2, utilizando FPGA Altera Max II Start Kit CPLD (*Complex Programmable Logic Device*). O foco do trabalho foi desenvolver uma interface UART com suporte a comunicação TII (*Transducer Independent Interface*), que é definida pelo padrão IEEE P1451.2. No trabalho, o autor definiu um novo nome para a interface, denominando-a UART-STIM (*Universal Asynchronous Receiver/Transmitter - Smart Transducer Interface Module*). Para o suporte de diferentes tipos de frequência, como por exemplo: UART, controlador, UFM (*User Flash Memory*) e conversor A/D (*Analogic/Digital*), foi utilizado um divisor de frequência. O MAX II Start Kit utilizado possui um oscilador de 18.432MHz e, com os divisores de 10 e 2560, foi possível obter 1.8432MHz para comunicação UART e 7.2KHz para UFM e conversores A/D. Para a comunicação do MAX II Starter Kit e o microcomputador, foi utilizado um circuito integrado denominado ICL3232 o qual possui as características do padrão RS-232. O conversor utilizado foi MCP3001 que, a partir de um conjunto de dados digitais, é passado para o STIM, utilizando uma interface SPI (*Serial Peripheral Interface Bus*). Como apresentação dos resultados, foi realizada uma comparação com a temperatura real e a medida pelo sistema, obtendo uma variação máxima de erro de 0.5 graus, porém, a taxa de *bytes* perdidos na transferência foi de zero por cento em 96 Bytes.

Higuera, Polo e Gasulla (2009) apresentaram em seu trabalho um NCAP (IEEE 1451.1) e um WTIM (IEEE 1451.5) utilizando a interface ZigBee. Os autores descreveram o módulo ZigBee (CC2420) utilizado para a comunicação e, de forma esquemática, a implementação do WTIM. Como parte do NCAP, foi descrita a interface desenvolvida em LabView, a qual inclui comandos de escrita e leitura para os transdutores, para os TEDS comandos de leitura, requisição e atualização.

No trabalho de Higuera, Polo e Gasulla (2009), é importante ressaltar a descrição das PHY TEDS para a interface sem fio ZigBee em que apresentaram todas as características relevantes da comunicação entre NCAP e WTIM, como por exemplo: protocolo de comunicação, frequência, distância máxima entre os módulos, tipo de antena, canais, quantidade de canais usados, especificação da bateria e modulação.

Em Manda e Gurkan (2009) apresentaram um *software* desenvolvido em .Net Framework para a descrição das TEDS. Os autores descreveram os campos dos TEDS,

as obrigatórias e as opcionais e o fluxograma do processo de conversão dos dados dos sensores em valores hexadecimal. Finalizando as características teóricas do sistema, os autores apresentaram o *software* e suas principais características, como o cadastro dos transdutores baseado na norma IEEE 1451.0-2007 e a conversão dos dados para hexadecimal. Uma característica que o autor descreve é a facilidade de conversão das informações em dados hexadecimais para armazenamento em memória.

Um trabalho que teve relevância para o desenvolvimento do nó de rede NCAP foi o de Lee e Song (2003) que desenvolveram uns dos primeiros modelos em UML (*Unified Modeling Language*) para o desenvolvimento do padrão IEEE 1451.1. Os autores descreveram as classes, modelo de objeto, a definição da classe NCAPBlock e a máquina de estado do NCAPBlock. Os autores descrevem também a facilidade de integração com outros modelos como: MIMOSA (*Machinery Information Management Open Systems Alliance*) e OSA-CBM (*Open System Architecture for Condition Based Maintenance*), sendo que, tais modelos, desde seu desenvolvimento foram baseados em diagrama de UML.

No trabalho de Stepanenko et al. (2006) apresentaram a hierarquia de classe do padrão IEEE 1451.1 através de diagrama. Além do diagrama de classe de todo o padrão IEEE 1451.1, os autores apresentaram também as quantidade de classes mínimas para o desenvolvimento do nó de rede NCAP através de diagrama e quais foram retirada do projeto do nó de rede. Para validação do modelo descrito, foi desenvolvido uma rede de sensores contendo vários NCAPs utilizando o microcontrolador Atmel AT89C51RC2 e a interface RS-232 para comunicação com o servidor. Uma contribuição do trabalho realizado, foi mostrar que é possível adaptar para qualquer interesse o padrão IEEE 1451.1 para os benefícios em redes distribuídas. Um outro fator, foi o desenvolvimento de um sistema utilizando dispositivos de baixo custo.

Cheng e Qin (2005) apresentaram o desenvolvimento de um NCAP e um STIM utilizando tecnologia Nios Soft-Core Processor. Os autores utilizaram o processador Nios para o desenvolvimento do NCAP e o STIM, o módulo LAN9IC111 para comunicação com a rede Ethernet e linguagem C para realização da programação. A comunicação entre o nó de rede NCAP e o STIM foi utilizada a interface TII, no qual os autores descrevem passo a passo a comunicação entre os módulos. Como conclusão, os autores destacam a flexibilidade, a configuração personalizada a reconfiguração e reprogramação.

Lee e Song (2007) apresentaram o desenvolvimento do nó de rede NCAP e o TIM utilizando notebooks. Para realizar a comunicação foi utilizada a interface sem fio padrão IEEE 802.11 e dois casos de estudos, o primeiro realizando a requisição de um sensor e

o segundo a leitura das TEDS. O WTIM realiza a comunicação com o microcontrolador utilizando a interface serial padrão RS-232 para que seja realizada a leitura do sensor. Para a demonstração dos testes, foram apresentados as interface usuários com as requisições e resposta do módulo WTIM, apresentando a leitura do sensor e a leitura das TEDS.

Nemeth-Johannes, Sweetser e Sweetser (2007) apresentaram o desenvolvimento do NCAP utilizando o microcomputador e o WTIM utilizando um processador ARM-32 bits. A comunicação entre os dispositivos foi feita utilizando o padrão IEEE 802.15.1 através do módulo de comunicação MCI *Module Communications Interface*. O TIM desenvolvido é denominado de TinyTIM e possui 6 canais de transdutores podendo ser expandido para mais canais.

O'Reilly et al. (2009) apresentaram as vantagens de utilizar um padrão aberto para redes de transdutores, aumentando a flexibilidade e aumentando a escalabilidade da rede. Apresentam-se também as camadas de protocolo, sendo que a camada mais alta encontra-se a Internet utilizada para aplicações remotas. Os autores fazem uma abordagem do padrão IEEE 1451 e especifica quais são as TEDS obrigatórias e as opcionais. O serviço web é baseado em SOA (*Service-Oriented Architecture*) e IEEE 1451.0. Os autores descrevem também a integração entre o padrão IEEE 1451 e OGC-SWE (*Open Geospatial Consortium - Sensor Web Enablement*), as especificações e os transdutores implementados pelo grupo para realização dos testes. Os autores também descrevem sobre a rede CAN (*Controller Area Network*), suas aplicações e a paralização do projeto em relação a integração com o padrão IEEE 1451.

Larrauri et al. (2010) apresentaram uma rede de transdutores para apoio em carros utilizando a interface Bluetooth e o padrão IEEE 1451. Para o desenvolvimento do nó de rede NCAP foi utilizado um microcomputador denominado mini-PC (alix 3c3) com o *hardware*: AMD Geode LX800, processador 500 MHz e 256 MB DDR DRAM (*Double Data Rate Dynamic Random-Access Memory*) com sistema operacional Linux implementado e o TIM foi utilizado microcontroladores da família Freescale HCS08. Os TIMs realiza a requisição dos dados e envia para NCAP, que processa os dados e envia através da rede Ethernet utilizando o protocolo HTTP (*Hypertext Transfer Protocol*). Os autores descrevem algumas desvantagens sobre o trabalho como, por exemplo, a implementação de apenas 2 canais TIMs no bloco NCAP e outra limitação que descrita é o limite máximo de TIMs, devido as características da rede Bluetooth.

Bodson (2010) apresentou uma breve descrição dos novos padrões definido pela IEEE, como por exemplo: WLAN 802.11P para comunicação em ambiente veicular, IEEE P1901 para indústria, IEEE 802.3BA para especificação de duas novas velocidades com novas

larguras de bandas para transmissão de dados em redes Ethernet e IEEE 1451.7, que faz parte da norma IEEE 1451 e utiliza interface e métodos para interfaceamento de transdutores para *tags* RFID.

Wang et al. (2005) apresentaram um sistema utilizando sensor de umidade e temperatura baseado no padrão IEEE P1451.2. Os dados são enviados para o NCAP utilizando o padrão IEEE P1451.2, que recebe a informação, processa e envia para um barramento local. O servidor recebe os dados e transmite para o cliente utilizando a Internet. Uma característica importante do trabalho, é a descrição das “CalibrationTEDS”, no qual os autores descrevem como foi feito a tabela do TEDS e como foi feito os cálculos baseados na tabela de calibração dos transdutores.

Song e Lee (2008a) descreveram as características do padrão IEEE 1451, quais os TEDS principais e uma pequena descrição de cada comitê. É importante destacar que, os autores descrevem o padrão IEEE P1451.6, porém, atualmente o projeto encontra-se parado. Os autores apresentam exemplo de projeto utilizando o padrão IEEE 1451.5 utilizando rede sem fio padrão IEEE 802.11. Os autores destacam também as aplicações do padrão, como, por exemplo: monitoramento e controle de forma remota, sistema de controle e medida distribuída e aplicações web.

Lee e Song (2005) apresentaram uma aplicação orientada a objetos utilizando camadas para o padrão IEEE 1451.4. Os autores dividiram o trabalho em 4 camadas: sistema operacional, utilitários, ferramentas, NCAP e aplicações. De forma sucinta, os autores descrevem sobre o padrão IEEE 1451, da arquitetura em camada do projeto desenvolvido e em que contexto se encaixa a estrutura de orientação a objetos. Foram apresentados também os diagramas em UML: hierarquia de classe do padrão IEEE 1451.1 e diagrama de agregação em relação as classes do padrão IEEE 1451.1. Como exemplo, foi utilizado sistema de tratamento de água para personalizar as camadas orientado a objeto.

Muitos trabalhos vêm sendo desenvolvidos baseado na norma IEEE 1451 utilizando diferentes arquiteturas e topologias de desenvolvimento. Os trabalhos relacionados nesta seção, apresentaram diferentes formas de desenvolvimento do NCAP e do TIMs, porém, neste trabalho apresenta-se um modelo novo de implementação do nó de rede NCAP, diferente dos modelos apresentados. O NCAP desenvolvido, possui arquitetura e sistema operacional embarcado dinâmico utilizando dispositivos reconfiguráveis (FPGA) baseando nas características do sistema em que será aplicado. O nó de rede NCAP é flexível e pode ser aplicado em diferentes ambientes, pois, a utilização da rede Ethernet como rede externa permite uma configuração simples e semelhante às utilizadas nos microcomputadores com sistema operacional Linux. Outra característica importante que facilita sua flexibilidade,

é a utilização de diferentes interfaces internas cabeada ou sem fio, baseada na norma IEEE 1451, ampliando o campo de aplicação.

1.5 Organização do Texto

Incluída a introdução geral no Capítulo 1 que contém a contextualização do projeto, um breve histórico dos sistemas operacionais embarcados e das redes de transdutores e os trabalhos mais relevantes na área, o texto foi organizado em 6 capítulos.

No Capítulo 2 são introduzidos os principais conceitos da família do padrão IEEE 1451 e a descrição de cada comitê, as tecnologias de transmissão de dados e como desenvolver os módulos NCAP e os TIMs.

No Capítulo 3 é apresentada uma descrição das interfaces de comunicação utilizadas para o desenvolvimento do nó de rede NCAP.

No Capítulo 4 são apresentadas as ferramentas de desenvolvimento e utilizadas como suporte, suas características, análise de desempenho e onde se enquadra dentro do contexto do projeto.

No Capítulo 5 são apresentadas as características de implementação dos TIMs e a descrição dos módulos desenvolvidos neste trabalho.

No Capítulo 6 é apresentado o desenvolvimento da parte lógica do *hardware* utilizando o processador Nios II, as configurações do sistema operacional μ Clinux e a inicialização do sistema e o *softwares* NCAP. Além das características de implementação são apresentados os testes e resultados obtidos pelo sistema.

No Capítulo 7 são apresentadas as conclusões gerais do trabalho e os trabalhos futuros a serem desenvolvidos.

Capítulo 2

O Padrão IEEE 1451

2.1 Introdução

O padrão IEEE 1451 tem como objetivo definir o interfaceamento do nó NCAP (IEEE 1451.1) e os diferentes módulos definidos de acordo com cada comitê: IEEE P1451.2, IEEE 1451.3, IEEE 1451.4, IEEE 1451.5, IEEE P1451.6 e IEEE 1451.7. Através do emprego do padrão IEEE 1451, é possível atingir interoperabilidade e o modo de funcionamento *plug and play* entre o NCAP e as diferentes tecnologias de interfaceamento, características definidas pelo padrão IEEE 1451.0. Nas Seções 2.2 a 2.9, serão apresentadas as definições estabelecidas em cada padrão da família IEEE 1451 e suas características.

2.2 O Padrão IEEE 1451.0 - 2007

O padrão IEEE 1451.0 - 2007 é um projeto que aponta uma funcionalidade comum, comandos e TEDS da família do padrão de transdutores inteligentes. Esta funcionalidade é independente do meio físico de comunicação. Isto inclui as funções básicas requeridas para controle e gerenciamento de transdutores inteligentes, protocolos de comando e independência do formato de mídia. O padrão IEEE 1451.0 especifica o formato dos TEDS que são “tabelas” de dados que contêm informações dos transdutores (sensores/atuadores) armazenadas em memória não volátil dentro do TIM. No entanto, há aplicações em que o armazenamento em uma memória não volátil não é prático para aplicação, então, o padrão IEEE 1451.0 permite o armazenamento em outros locais de forma remota denominando-as de TEDS Virtuais. Os TEDS Virtuais são arquivos eletrônicos que fornecem as mesmas funcionalidades das implementadas em memória no TIM, porém, não estão no TIM (IEEE, 2007c).

Uma técnica para atingir a característica *plug-and-play* foi a definição de um conjunto mínimo de informações dos transdutores e outros recursos opcionais para funções mais avançadas. Os TIMs, ao serem conectados ao NCAP, transmitem as informações dos TEDS ao gerenciador de protocolo, que faz o reconhecimento da rede de transdutores de forma automática, ou seja, *plug and play* (IEEE, 2007c). Para descrever os TEDS, o padrão IEEE 1451.0 - 2007 apresenta um formato denominado de TLV (*Type/Length/Value*) que é descrito na Seção 2.2.1.

A característica *plug and play* dos transdutores inteligentes aos seus usuários e desenvolvedores faz surgir algumas vantagens como: redução do tempo para parametrização do sistema, diagnósticos avançados, redução do tempo para reparo e reposição, gerenciamento avançado do *hardware* e automatização da calibração (IEEE, 2007c).

Para a implementação dos TEDS, quatro blocos são obrigatórios, Meta-TEDS, TransducerChannel TEDS, User's Transducer Name TEDS e PHY TEDS. As demais tabelas, Calibration TEDS, Frequency Response TEDS, Transfer Function TEDS, Text-based TEDS, Commands TEDS, Identification TEDS, Geographic location TEDS, Units extension TEDS, End User Application Specific TEDS e Manufacturer defined TEDS são opcionais. Neste trabalho, foram utilizadas os TEDS obrigatórias para testes do sistema e implementação do padrão IEEE 1451.0-2007.

A Meta-TEDS fornece uma identificação única denominada UUID (*Universal Unique Identifier*) e alguns parâmetros de tempo que são disponíveis para serem usadas pelo NCAP, como por exemplo, o tempo excedido na comunicação de um determinado *software* quando o TIM não está respondendo. O restante dos campos dos TEDS descreve o relacionamento entre o TransducerChannel que possui o TIM. No Apêndice C, apresentam-se exemplos das Meta-TEDS descritos nos TIMS desenvolvidos neste trabalho.

O TransducerChannel TEDS deve conter as informações detalhadas sobre as especificações do transdutor, fornecendo parâmetro físico que está sendo medido ou controlado, a faixa no qual o transdutor opera, as características de I/O digital, modo de operação da unidade e especificações de tempo. No Apêndice C, apresentam-se exemplos das TransducerChannel TEDS descritos nos TIMS desenvolvidos neste trabalho.

O User's Transducer Name TEDS contém o nome do TIM ou canal do transdutor para que seja reconhecido no nó de rede NCAP. A estrutura desta TEDS é recomendada no padrão, mas o tamanho do conteúdo é definido pelo usuário. No Apêndice C apresentam-se exemplos das User's Transducer TEDS descritos nos TIMS desenvolvidos neste trabalho.

A PHY TEDS é dependente do meio de comunicação físico usado para conectar o NCAP e o TIM. No Apêndice C, apresentam-se exemplos das PHY TEDS descritos nos TIMS desenvolvidos neste trabalho.

O padrão IEEE 1451.0, além de definir padrões para armazenamento das informações dos transdutores inteligentes, também fornece normas para mensagens (*Message structures*) e comando (*Command*). *Message structures* definem a estrutura das mensagens enviadas através da interface de comunicação. *Command* define as tarefas que serão executadas como estado ou operação que será realizada (IEEE, 2007c). Nas Seções 2.2.2 e 2.2.3 apresentam-se exemplos de estrutura de mensagens e comandos enviados pela interface entre o NCAP e o TIM.

2.2.1 Formato padrão dos TEDS

Os TEDS possuem um formato padrão apresentado na Tabela 1. O primeiro campo em quaisquer TEDS é o tamanho e é representado por 4 octetos sem sinal inteiro. O próximo bloco representa o conteúdo dos TEDS, podendo ser representado por informações binárias ou baseadas em texto. O último campo em quaisquer TEDS é o *checksum*, que é usado para verificar a integridade dos dados dos TEDS.

Tabela 1 – Formato padrão dos TEDS.

Campo	Descrição	Tipo	Octeto
—	Tamanho dos TEDS	UInt	4
1 para N	Bloco de dados	Variável	Variável
—	Checksum	UInt16	2

Fonte: IEEE (2007c)

Os campos da tabela TEDS são descritos como:

- **Tamanho dos TEDS** - é o número total de octetos no bloco de dados mais os dois octetos do *checksum*.
- **Bloco de dados** - campo que contém informações específicas de acordo com cada tabela TEDS. Os campos que compõem esta estrutura são diferentes para cada tipo de TEDS e a sua estrutura é baseada em TLV, exceto o padrão IEEE 1451.2-1997 e IEEE 1451.3-2003 (IEEE, 2007c). A construção de cada linha dentro da tabela TEDS utiliza a estrutura TLV definida como:
 - **Type** - campo definido por 1 octeto, representa a identificação da linha TLV;
 - **Length** - especifica o número de octetos no campo *Value*;

- **Value** - contém as informações do campo TEDS;
- **Checksum** - é o complemento da soma de todos os octetos precedidos incluindo o campo inicial tamanho dos TEDS.

2.2.2 Estrutura das Mensagens

Os dados utilizados na transmissão entre o NCAP e o TIM são descritos em nível de octeto. Na Tabela 2, apresenta-se um grupo de octetos e a ordem de transmissão em que é definido pelo padrão IEEE 1451.0.

Tabela 2 – Ordem de transmissão dos dados.

1-Octeto	1-Octeto	1-Octeto	1-Octeto
8º bit ... 1º bit	8º bit ... 1º bit	8º bit ... 1º bit	8º bit ... 1º bit
1º octeto transmitido	2º	3º	4º
5º	6º	7º	8º

Fonte: IEEE (2007c)

Para representar uma quantidade numérica, o bit mais à esquerda é representado como sendo o mais significativo e, o mais à direita, como sendo o menos significativo. Na Tabela 3 apresenta-se o valor numérico e a ordem de transmissão de dados, de acordo com o bit mais significativo e o menos significativo.

Tabela 3 – Exemplo de bit mais significativo e menos significativo.

Ordem	7	6	5	4	3	2	1	0
Valor = 170	1	0	1	0	1	0	1	0

Fonte: IEEE (2007c)

A estrutura de uma mensagem de comando de requisição é definida de acordo com a Tabela 4 e é descrita como:

- **Número do canal do transdutor** - campo representado por 16 bits e define o canal do transdutor de destino;
- **Classe de comando** - campo representado por 8 bits e define a classe do comando de acordo com a Tabela 5;
- **Função do comando** - campo representado por 8 bits que define a função do comando de acordo com o que foi atribuído na classe de comando. Na Tabela 6 apresentam-se as funções de operação do transdutor baseado na classe de comando representado pelo atributo: XdcrOperate utilizada neste projeto;

- **Tamanho** - campo representado por 16 bits em que define o tamanho do campo de dados. O tamanho de uma mensagem recebida, se não relacionar com o campo tamanho da mensagem, esta é descartada e uma mensagem de erro é retornada;
- **Campo de dados** - contém as informações a serem transmitidas.

Tabela 4 – Estrutura de uma mensagem de comando

1 - Octeto							
7	6	5	4	3	2	1	0
Número do canal do transdutor (octeto mais significativo)							
Número do canal do transdutor (octeto menos significativo)							
Classe de comando							
Função do comando							
Tamanho (octeto mais significativo)							
Tamanho (octeto menos significativo)							
Campo de dados							

Fonte: IEEE (2007c)

Tabela 5 – Classe de comando.

Identificação do comando	Nome do atributo	Categoria
0	Reserved	Reservado
1	CommonCmd	Comando comum para o TIM e o canal do transdutor
2	XdcrIdle	Estado ocioso do transdutor
3	XdcrOperate	Estado de operação do transdutor
4	XdcrEither	Transdutor em estado ocioso ou operando
5	TIMsleep	Estado de dormência
6	TIMActive	Comando de estado ativo do TIM
7	AnyState	Qualquer estado
8-127	ReservedClass	Reservada
127-255	ClassN	Aberto para os fabricantes

Fonte: IEEE (2007c)

A estrutura de uma mensagem de comando de resposta é definida de acordo com a Tabela 7 e é descrita como:

- **Sucesso/Falha** - campo representado por 8 bits e define se os dados foram enviados com sucesso ou não. Caso o valor do campo for diferente de zero, a mensagem foi enviada com sucesso, senão, é considerada uma mensagem falha e o sistema pode verificar a mensagem para identificar o problema;
- **Tamanho** - campo representado por 16 bits e define o tamanho do campo de dados. O tamanho de uma mensagem recebida, se não relacionar com o campo tamanho da mensagem, a mensagem é descartada;

Tabela 6 – Comando de operação do transdutor.

Identificação	Comando	Canal de transdutor	Global/Grupo
0	Reservado	X	—
1	Ler um canal de transdutor	X	—
2	Escrever um canal de transdutor	X	—
3	Comando <i>Trigger</i>	X	X
4	Abortar <i>Trigger</i>	X	X
5-127	Reservado	—	—
127-255	Aberto para fabricantes	—	—

Fonte: IEEE (2007c)

- **Campo de dados** - contém as informações a serem transmitidas.

Tabela 7 – Mensagem de comando de resposta.

1 - Octeto							
7	6	5	4	3	2	1	0
Sucesso/Falha							
Tamanho (octeto mais significativo)							
Tamanho (octeto menos significativo)							
Campo de dados							

Fonte: IEEE (2007c)

2.2.3 Comandos

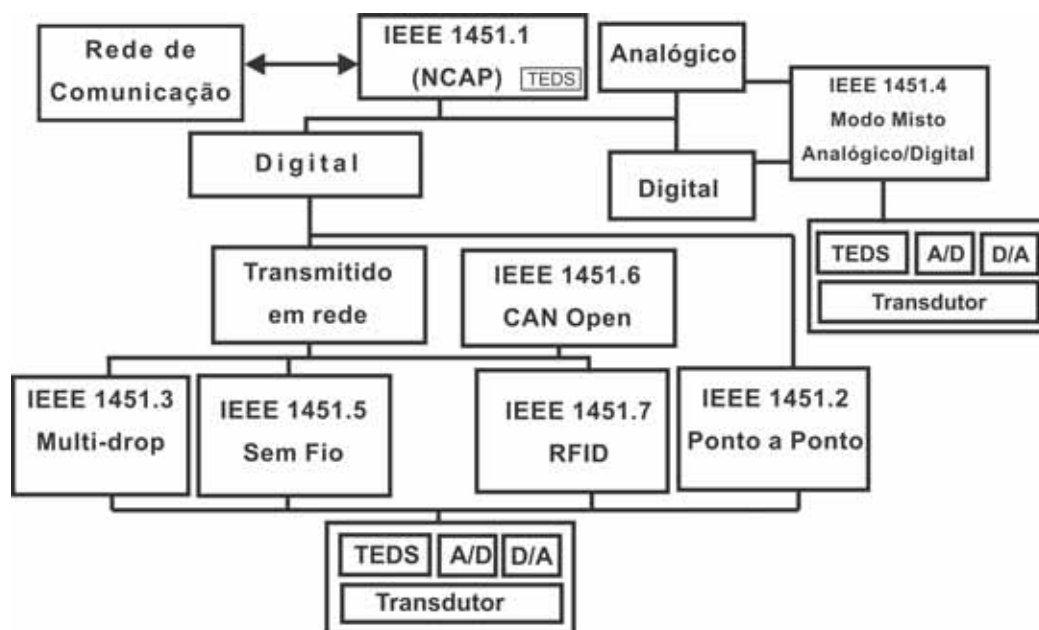
Os comandos definidos pelo padrão IEEE 1451.0 são divididos em duas categorias: padrão (definido pelo padrão) e fabricante (definido pelo fabricante). Independente da categoria, o comando é dividido em dois octetos, o mais significativo que identifica a classe de comando e o menos significativo que identifica a função do comando, como apresentado na Tabela 4. Na categoria padrão, a classe e as funções são definidas de acordo com a norma IEEE 1451.0. Um exemplo é que se o octeto mais significativo que especifica a classe definir estado de operação do transdutor (atributo: XdcrOperate) apresentado na Tabela 5, o octeto menos significativo que especifica a função deve ser definido de acordo com uma das especificações definidas na Tabela 6. Na categoria fabricante, as especificações do comando não são definidas pelo padrão IEEE 1451.0, mas pelo usuário que está desenvolvendo o sistema, tornando-se a responsabilidade do usuário de desenvolver um *software* para interpretar os comandos. No Apêndice B apresenta-se exemplos de comandos utilizados para testes do sistema.

2.3 O Padrão IEEE 1451.1

A fabricação tradicional de transdutores tem evidenciado um grande esforço no desenvolvimento do interfaceamento de sensores e atuadores para as redes de controle. Usualmente, o resultado do esforço do desenvolvimento de uma rede pode não ser fácil transportar para outras redes. O principal objetivo da NIST/IEEE é o desenvolvimento de métodos de conexão padronizado entre transdutores inteligentes e redes de controle utilizando tecnologias já existentes. O padrão IEEE 1451.1 especifica a interface com a rede de tal maneira que a aplicação independa do protocolo utilizado pela rede ou barramento de campo (IEEE, 1999).

Um NCAP em conformidade com o padrão IEEE 1451.1 pode funcionar com TIMs utilizando diferentes interfaces baseando-se nas normas de interfaceamento do padrão IEEE 1451. Na Figura 2 é ilustrada a família do padrão IEEE 1451 e suas possíveis conexões. O padrão IEEE 1451.0 é representado através do bloco TEDS.

Figura 2 – Diagrama da família do padrão IEEE 1451.



Fonte: Santos Filho (2012)

Na Seção 2.3.1 apresenta-se as especificações do nó de rede NCAP baseadas na norma IEEE 1451.1.

2.3.1 Processador de Aplicação com Capacidade de Operar em Rede (NCAP) - IEEE 1451.1

O NCAP é um nó com capacidade de processamento local, capaz de receber dados de uma rede externa através de um determinado protocolo, converter essas informações para outro tipo de protocolo baseado na norma IEEE 1451 para realizar a comunicação com a rede de transdutores, serviço semelhante a um *gateway*¹. Para isto, o nó NCAP tem como característica identificar o tipo de rede no qual está conectado, dando assim o conceito de interoperabilidade, além de abstrair informações do tipo de transdutor conectado, facilitando a introdução do modo de operação *plug and play* (IEEE, 1999).

Para o seu desenvolvimento, o nó de rede NCAP é subdividido em 2 partes, física e lógica. A parte física é composta por microprocessador e seus circuitos associados, *hardware* de comunicação com a rede e as interfaces de acesso I/O. As características físicas do *hardware* não são definidas pelo padrão IEEE 1451. A parte lógica compreende os componentes lógicos que podem ser agrupados em componentes de aplicação e suporte. Os componentes de suporte são o sistema operacional, o protocolo da rede e a *firmware*² dos transdutores inteligentes. O sistema operacional tem como objetivo realizar o interfaceamento entre as aplicações e o *hardware*, facilitando a manipulação dos dados e o acesso às interfaces (IEEE, 1999).

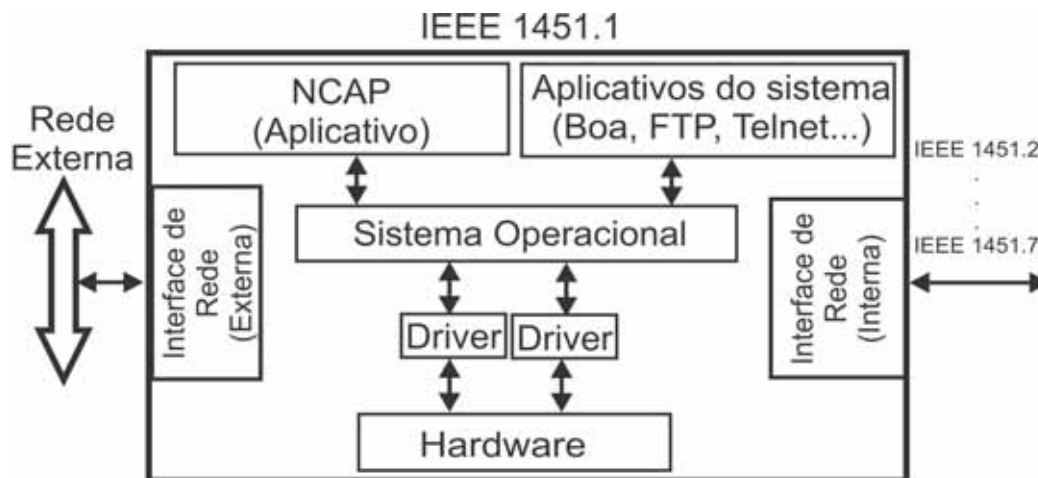
Na Figura 3 apresenta-se um modelo de nó de rede contendo um processador, *driver* para a comunicação do sistema operacional com os periféricos, o sistema operacional realizando o gerenciamento dos arquivos e o interfaceamento dos aplicativos com o *hardware*. Em nível de abstração mais alto, apresentado na Figura 3, o nó de rede é responsável por realizar a aquisição dos dados de uma rede de comunicação externa, processar esses dados e realizar o controle ou o monitoramento de acordo com umas das normas da família do padrão IEEE 1451.

Um NCAP, em conformidade com o padrão IEEE 1451.1, pode operar com o STIM através de uma interface TII, RS-232, USB, SPI e I2C (IEEE P1451.2), o TBIM (*Transducer Bus Interface Module*) através de um barramento (IEEE 1451.3), o MMI (*Mixed-Mode Interface*) através de uma interface com sinais mistos (IEEE 1451.4), o WTIM através da comunicação sem fio (IEEE 1451.5), o TBC (*Transducer Bus Controller*) utilizando um barramento CAN (IEEE P1451.6) ou RFID (*Radio-Frequency Identification*) através de sinais de rádio frequência (IEEE 1451.7).

¹*Gateway*- é um dispositivo intermediário geralmente destinado a interligar redes, separar domínios de colisão, ou mesmo traduzir protocolos.

²Software responsável pela inicialização de um processador e seus circuitos de interface

Figura 3 – Bloco NCAP genérico com as principais descrições.



Fonte: Santos Filho (2012)

O padrão IEEE 1451.1 especifica-se uma arquitetura de *software* aplicável em sistemas distribuídos, podendo ter um ou mais NCAPs se comunicando em rede. A especificação da arquitetura de *software* é apresentada na documentação do padrão IEEE 1451.1 como sendo orientada a objetos. Entretanto, não há a exigência do uso de técnicas de implementação orientado a objetos (IEEE, 1999). A arquitetura do *software* do NCAP é baseada em: modelo de informação, modelos de dados e modelos de comunicação em rede. Nas Seções 2.3.1.1, 2.3.1.2 e 2.3.1.3 apresentam-se a descrição de cada modelo.

2.3.1.1 Modelo de Informação

O modelo de informação é composto de uma hierarquia de classes de objetos dividido em três principais categorias:

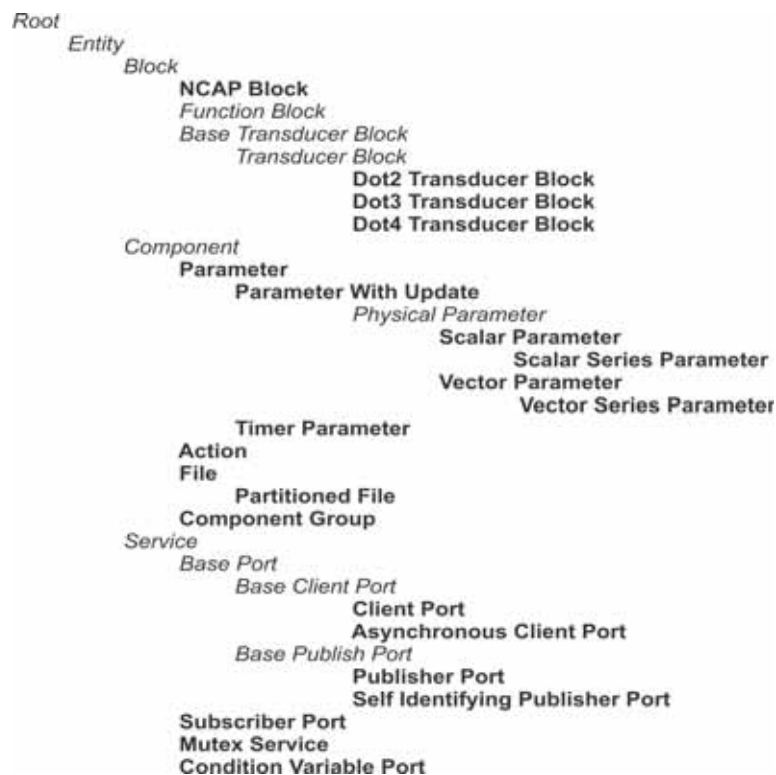
- **Classes *Block*** - realiza o processamento;
- **Classes *Component*** - encapsula os dados;
- **Classes *Service*** - comunicação do NCAP e sincronização das interfaces de comunicação.

Na Figura 4 apresenta-se a hierarquia de classes do padrão IEEE 1451.1.

Uma importante característica na implementação do NCAP é a forma de descrição dos nomes das classes, métodos e declaração das variáveis, como por exemplo:

- **Nome das classes** - IEEE1451_ FunctionBlock (sem separação da palavra inicial

Figura 4 – Hierarquia de classes do padrão IEEE 1451.



Fonte: IEEE (1999)

e cada letra inicial da palavra em maiúsculo, prefixo IEEE1451_ NomeDaClasse se especificada pelo padrão IEEE 1451.1, fonte do computador);

- **Métodos** - TemperatureSensor (sem separação da palavra, inicial de cada palavra em maiúsculo e fonte do computador);
- **Variáveis** - read, read_status (separação com sublinhado, sem letra maiúscula, fonte do computador).

2.3.1.2 Modelo de Dados

O modelo de dados define o tipo de dados usado para trocar informação entre as entidades dentro de um processo do NCAP e através da rede. O modelo de dados suporta os seguintes tipos de dados:

- **Boolean** - *True* ou *False*;
- **Octet** - 8 bits não interpretado como um número;
- **Integer** - número com sinal ou sem sinal de 8 bits a 64 bits;

- **Float** - numérico baseado no padrão IEEE 754;
- **String** - *Array* de octetos contendo texto de leitura humana.

2.3.1.3 Modelos de Comunicação em Rede

O padrão IEEE 1451 provê dois modelos de comunicação entre os objetos em um sistema distribuído: cliente/servidor e editor/subscritor (IEEE, 1999).

No modelo cliente/servidor, os usuários denominados clientes realizam uma solicitação através de mensagens para o servidor e de acordo com a mensagem, é executado um determinado processo. Ao finalizar o processo, o servidor envia uma mensagem de resposta ao cliente.

No modelo editor/subscritor, o emissor (editor) não necessita de enumerar explicitamente os receptores (subscritor) das mensagens. Quando o editor envia uma mensagem, o sistema de editor/subscritor é responsável pelo correto encaminhamento dos dados até todos os subscritores que desejam receber. De modo a receber a mensagem o subscritor pode expressar que categoria de mensagem deseja receber, ou definir o conjunto de características que a mensagem deve possuir para que possa estar recebendo. Um exemplo de sistema distribuído editor/subscritor bem conhecido é *BitTorrent*, no qual várias pessoas estão interligadas através de um sistema de rastreio que coordena o encaminhamento das informações (IEEE, 1999).

É importante destacar que neste trabalho foi implementado o modelo cliente/servidor para o desenvolvimento do nó de rede NCAP. Nas Seções 2.4 a 2.9, apresentam-se as normas de interfaceamento entre o NCAP e os TIMs definidas pelo padrão IEEE 1451.

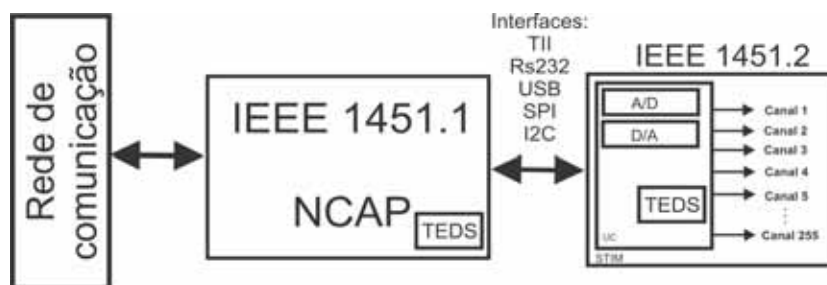
2.4 O Padrão IEEE P1451.2

O padrão IEEE P1451.2 introduz o conceito de STIM (*Smart Transducer Interface Module*). O STIM é um módulo que contém um ou mais transdutores conectados, constituindo uma aplicação de transdutores que tem como objetivo fornecer capacidade *plug and play* para conectar transdutores em rede, facilitar o suporte aos clientes, fornecer um conjunto mínimo de ferramentas que possibilite a identificação dos transdutores e fazer com que os sistemas implementados sejam escaláveis.

Um STIM pode ser composto de 1 até 255 transdutores por módulo e cada módulo pode conter circuitos de condicionamento de sinal, conversores A/D (analogico/digital) e D/A (digital/analogico), uma memória não volátil para o armazenamento dos TEDS

e lógica necessária para realizar a comunicação através de uma de suas interfaces padronizadas: TII, RS-232, SPI, I2C ou USB. É importante destacar que o padrão IEEE P1451.2-1997 não é compatível com o padrão IEEE 1451.0 - 2007, mas o grupo de trabalho do IEEE P1451.2 está realizando as revisões e fazendo com que a norma torne compatível com o padrão IEEE 1451.0 (SONG; LEE, 2008b) . Em Song e Lee (2008b) apresenta-se uma proposta de desenvolvimento do padrão IEEE 1451.0 com IEEE P1451.2 que foi utilizado como referência para o desenvolvimento do STIM neste trabalho. Na Figura 5 é ilustrado um modelo de STIM, suas interfaces e um exemplo de configuração de *hardware* para o seu desenvolvimento.

Figura 5 – Modelo do padrão IEEE P1451.2.

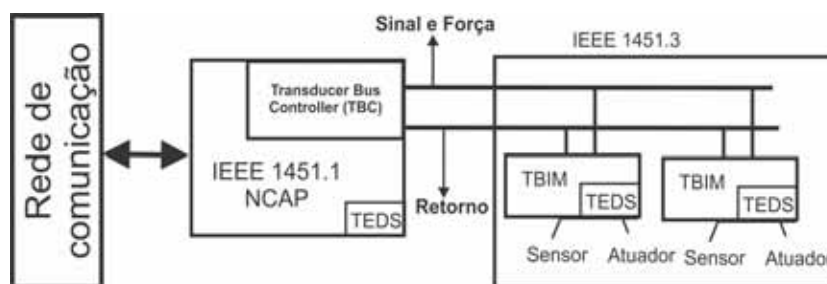


Fonte: Santos Filho (2012)

2.5 O Padrão IEEE 1451.3

O padrão IEEE 1451.3 foi desenvolvido com o objetivo de criar uma norma padrão de interfaceamento de transdutores em rede distribuída, utilizando o barramento *multdrop*. O padrão introduz o conceito de TBIM (*Transducer Bus Interface Module*) e um TBC conectado por um barramento de transdutores. Um TBIM é um módulo que contém o barramento de interface, condicionamento de sinal, conversor A/D ou D/A e os transdutores necessários. O TBC é um bloco que contém elementos de *hardware* e *software* no NCAP ou um *host* que provê o interfaceamento para o barramento de transdutores. O barramento de transdutores provê uma comunicação entre um NCAP, *host* ou mais TBIMs (LEE, 2009). Como apresentado na Figura 6, uma única linha de transmissão é utilizada para fornecer energia aos transdutores e fornecer a comunicação entre o controlador e os TBIMs. Em um único barramento, espera-se controlar no máximo 255 módulos TBIMs e, em cada módulo TBIM, um total de 255 sensores ou atuadores, chegando a um total de 65025 transdutores em cada barramento (NIST, 2011).

Figura 6 – Modelo do padrão IEEE 1451.3.

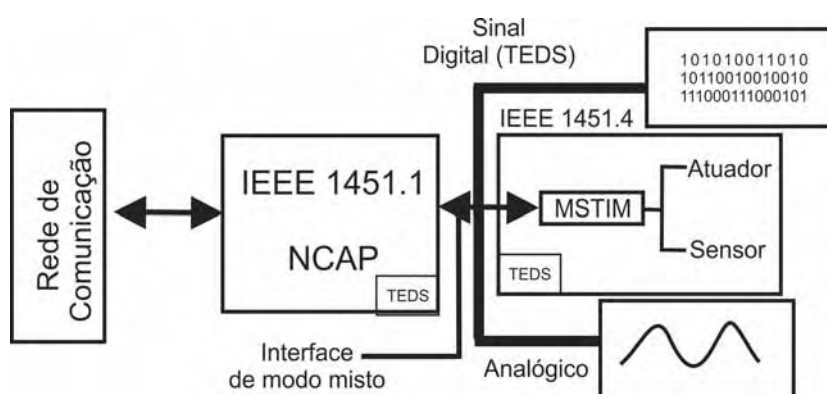


Fonte: Santos Filho (2012)

2.6 O padrão IEEE 1451.4

O padrão IEEE 1451.4 tem como principal foco prover a compatibilidade com o MMI de forma analógico e digital. A interface transmite os dados das TEDS através de um canal digital e as informações dos sensores utilizando o canal analógico, dando, assim, a capacidade de *plug and play* aos módulos MMI. Para realizar a comunicação entre o MMI e NCAP, é realizada a comutação entre a memória de armazenamento dos TEDS e a saída do transdutor, sendo que ambos compartilham um único meio físico. Na Figura 7 apresenta-se o modelo de interfaceamento de acordo com a norma IEEE 1451.4 que tem por objetivo realizar o interfaceamento de transdutores analógicos e transformá-los em módulos de transdutores *plug and play*, tornar o mais simples possível a implementação de transdutores inteligentes e garantir a flexibilidade dos MMI, podendo conectá-los com outras redes (IEEE, 2004).

Figura 7 – Modelo do padrão IEEE 1451.4.



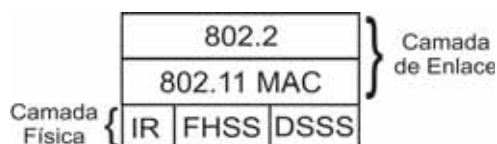
Fonte: Santos Filho (2012)

2.7 O padrão IEEE 1451.5

O padrão IEEE 1451.5 tem como objetivo especificar o interfaceamento do nó NCAP com o módulo WTIM (*Wireless Transducer Interface Module*), de acordo com uma das tecnologias de rede sem fio: Wi-Fi (*Wireless Fidelity*) (padrão IEEE 802.11) (TANENBAUM; WETHERALL, 2011), ZigBee (padrão IEEE 802.15.4) (FARAHANI, 2008) ou Bluetooth (padrão IEEE 802.15.1) (TANENBAUM; WETHERALL, 2011).

O IEEE 802.11 é um padrão que foi desenvolvido em 1997 por um grupo de trabalho do IEEE para definir o conceito de redes locais sem fio, a WLAN (*Wireless Local Area Network*). A família 802.11 é subdividida em outras 4 especificações: IEEE 802.11, 802.11b, 802.11a, 802.11g. Além disso, existe previsão para um novo padrão definido como IEEE 802.11i. O padrão IEEE 802.11 cobre as especificações técnicas das camadas de enlace e física, tendo como referência a hierarquia das camadas do modelo OSI (*Open Systems Interconnection*). Os três meios de transmissão de dados da camada física são FHSS (*Frequency-Hopping Spread Spectrum*), DSSS (*Direct-Sequence Spread Spectrum*) e o IR (*Infra Red* - Infravermelho). Os padrões FHSS e DSSS trabalham com *Spread Spectrum* de sinal e operam em uma frequência de 2,4 MHz, sendo que ambas trabalham com faixa de transmissão de 1 a 2 Mbps. A comunicação serial IRDA (*Infrared Data Association*) suporta transmissão de dados assíncrona em uma taxa de 115,200 bps utilizando *half-duplex* (AXELSON, 2007). A comunicação é realizada bit a bit semelhante a comunicação serial RS-232, sendo que inicia-se com 1 bit de start, seguindo de 1 Byte de dados, 1 bit de paridade, encerrando a transferência do pacote com 1 bit de parada. Na Figura 8, apresentam-se as 2 camadas utilizadas nas redes Wi-Fi. O padrão IEEE 802.2, definido na Figura 8, especifica o LLC (*Logical Link Control*) responsável por implementar a interface do nível de enlace com o nível de rede, fornecer serviços como multiplexação e o controle do fluxo e dos erros (KUROSE; ROSS, 2005).

Figura 8 – Camada de enlace e camada física.



Fonte:Kurose e Ross (2005)

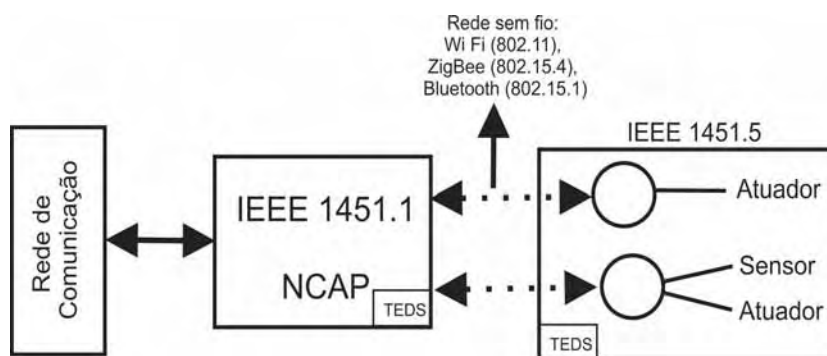
Na Seção 3.4 apresenta-se com detalhes a tecnologia ZigBee que foi utilizada neste trabalho para realizar a comunicação entre o NCAP e o WTIM.

A rede Bluetooth é uma especificação para a comunicação em curta distância de redes sem fio, com um baixo custo e alta operabilidade. Seu protocolo trabalha sobre a faixa

de frequência definido pela ISM (*Industrial, Scientific and Medical*), operando em 2,45 GHz com uma taxa de transferência de até 723,1 Kbps (KOBAYASHI, 2004). Para evitar interferência com outros protocolos de redes sem fio, a tecnologia Bluetooth possui 79 canais e podem ser chaveadas até 1600 vezes por segundo. Apesar de trabalhar na mesma frequência das redes Wi-Fi, a rede Bluetooth é mais lenta e suporta poucos dispositivos, no entanto, uma de suas vantagens é fornecer uma topologia dinâmica chamada de piconet ou PAN (*Personal Area Network*) contendo, no mínimo, dois e, no máximo, oito nós em cada rede (KUROSE; ROSS, 2005).

Na Figura 9 apresetam-se os padrões sem fio utilizados para realizar o interfaceamento do NCAP e WTIM.

Figura 9 – Modelo do padrão IEEE 1451.5.



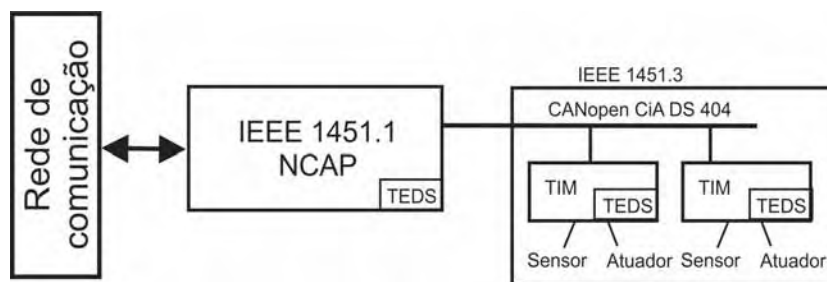
Fonte: Santos Filho (2012)

2.8 O padrão IEEE P1451.6

O padrão IEEE P1451.6 define uma interface de transdutores inteligentes entre o NCAP e TIM utilizando a rede CANopen de alta velocidade. A rede CANopen define o mapeamento do IEEE P1451.6 para dicionários dos TEDS, parâmetro de configuração, processamento de dados e informações de diagnóstico. Existem diferentes implementações de interfaceamento da rede CANopen, baseadas no CiA DS 404 (*CAN in Automation - Device Specification 404*), buscando utilizar os conceitos de IEEE 1451. A norma proposta irá permitir o desenvolvimento de *gateways* e redes de transdutores em cascata, que combinam especificações do IEEE 1451 e CANopen (IEEE, 2007a). É importante ressaltar que o grupo do padrão IEEE P1451.6 não se encontra ativo e a documentação não está finalizada (REILLY et al., 2010).

Na Figura 10 é ilustrado o modelo de rede CANopen, de acordo com sua especificação no CiA DS 404, o interfaceamento com o módulo NCAP e os diferentes TIMs.

Figura 10 – Modelo do padrão IEEE P1451.6.

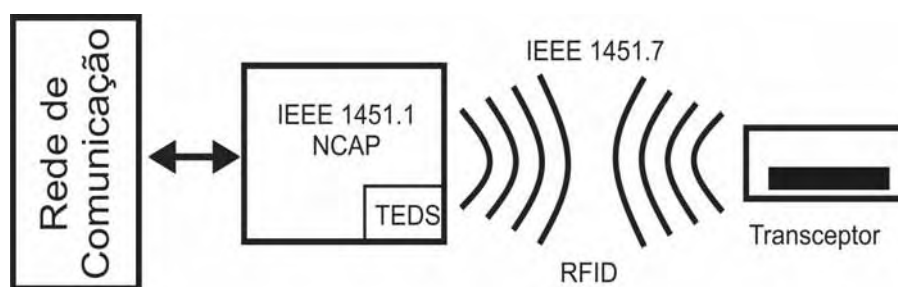


Fonte: Santos Filho (2012)

2.9 O padrão IEEE 1451.7

O padrão IEEE 1451.7 propõe o interfaceamento do NCAP e o TIM através da transmissão RFID que se refere ao padrão ISO/IEC 24753 (Organização Internacional para Padronização / Comissão Eletrotécnica Internacional). O RFID é uma tecnologia que foi criada durante a II Guerra Mundial por Sir Robert Alexander Watson-Watt, um físico escocês, que utilizava sua descoberta para identificação de aviões inimigos. Diferente de sua origem, hoje, o RFID é utilizado para fins pacíficos aplicados em diversas áreas, como: industrial, comercial, bibliotecas, segurança, identificação de animais etc. O padrão IEEE 1451.7 tem como característica a transmissão dos dados dos transdutores e dos TEDS, utilizando RFID (IEEE, 2010). Na Figura 11 apresenta-se um exemplo de modelo referente ao padrão IEEE 1451.7.

Figura 11 – Modelo do padrão IEEE 1451.7.



Fonte: Santos Filho (2012)

2.10 Considerações Finais Sobre o Capítulo 2

Neste capítulo foram apresentadas as características de cada comitê definido de acordo com o padrão IEEE 1451. É importante ressaltar que o nó de rede NCAP desenvolvido neste trabalho foi baseado nas normas IEEE 1451.0, IEEE 1451.1, IEEE P1451.2 (RS-232) e IEEE 1451.5 (ZigBee), foram apresentadas as suas características e as formas de

implementação baseadas em cada norma. As normas IEEE 1451.3, IEEE 1451.4, IEEE P1451.6 e IEEE 1451.7 foram apresentadas de forma sucinta mostrando as principais características.

Capítulo 3

Interfaces de Comunicação Utilizadas no Nó de Rede IEEE 1451

3.1 Introdução

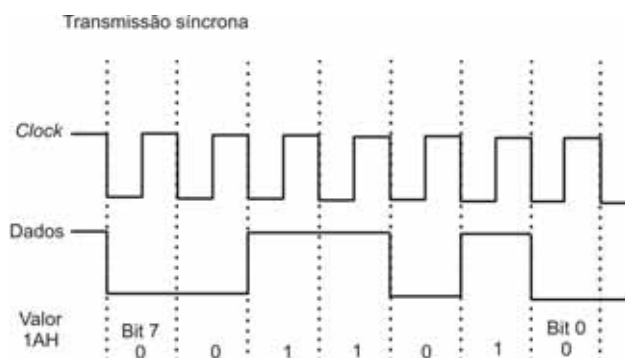
Neste capítulo apresenta-se as características das interfaces de comunicação e os padrões de interfaceamento utilizados para o desenvolvimento do nó de rede NCAP. Na Seção 3.2 apresenta-se de forma sucinta uma abordagem de transmissão síncrona e assíncrona, *bit rate e baud rate* e o padrão RS-232 utilizado para realizar um dos interfaceamentos entre o nó de rede NCAP e o TIM. Na Seção 3.3 apresenta-se a interface SPI, utilizada para realizar a comunicação com o cartão de memória SD Card. Na Seção 3.4 apresentam-se características da rede ZigBee, utilizada para realizar o interfaceamento entre o NCAP e os WTIMs. Na Seção 3.5 são apresentadas as principais características da rede Ethernet, utilizada neste trabalho como rede externa ao nó de rede NCAP.

3.2 Comunicação Serial

Comunicação serial é o processo de enviar dados bit a bit, sequencialmente, em um canal de comunicação ou barramento. A comunicação trabalha com uma porta de saída serial como sendo um transmissor que envia bits em um tempo pré-determinado para uma porta serial de entrada que funciona como receptor. O cabo entre os terminais tipicamente possui caminho de dados dedicado para cada direção, sendo, Tx representando a transmissão e Rx a recepção. A comunicação serial entre os dispositivos é definida de duas formas: síncrona e assíncrona. Na comunicação síncrona, a interface inclui um barramento para o *clock* tipicamente controlada por um dos dispositivos e todos os bits

transmitidos são sincronizados com o *clock*. Cada bit transmitido é válido em um tempo definido de acordo com o *clock*, podendo ser ativado de acordo com a borda de subida ou descida, dependendo do protocolo. Exemplos de comunicação síncrona são: I2C, SPI e Microwire (AXELSON, 2007) . Na Figura 12 apresenta-se um exemplo de transmissão síncrona com um barramento de *clock* e um de dados.

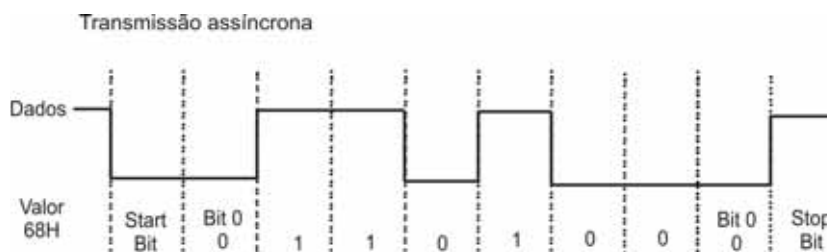
Figura 12 – Transmissão serial síncrona.



Fonte: Axelson (2007)

Na comunicação assíncrona, a interface não inclui um barramento para o *clock*. Dessa forma, para realizar a comunicação, cada dispositivo utiliza seu próprio *clock* para uma referência do tempo. Ambos os dispositivos devem possuir a mesma frequência e as frequências reais em cada dispositivo devem corresponder dentro de um pequeno percentual. Para inicializar a comunicação, um *Start* bit realiza o sincronismo do *clock* transmissor e receptor e, para finalizar, um *Stop* bit. Na Figura 13, apresenta-se um exemplo de transmissão assíncrona (AXELSON, 2007).

Figura 13 – Transmissão serial assíncrona.



Fonte: Axelson (2007)

Uma UART (*Universal Asynchronous Receiver Transmitter*) transmite os dados em partes, frequentemente chamados de palavras. Cada palavra contém um *Start* bit, bits de dados, um bit de paridade (opcional) e um ou mais *Stop* bits, a maioria das UART suportam múltiplos formatos. Um formato comum é 8-N-1, em que cada palavra transmitida possui um *Start* bit, seguido por 8 bits de dados e um *Stop* bit. O N indica que a palavra não possui um bit de paridade. As palavras que possuem o bit de paridade, são

definidas de acordo com os seguintes tipos: *even*, *odd*, *mark* ou *space*. O bit de paridade pode prover uma forma simples de detecção de erros, com paridade em *even*, os bits de dados e o bit de paridade, em cada palavra, contêm um número par de 1s. Com paridade *odd*, os bits de dados e o bit de paridade em cada palavra contêm um número ímpar de 1s. *Mark* e *space* são formas para trabalhar com o bit de paridade. Com a paridade *mark*, o bit de paridade é sempre 1s e, com a paridade *space*, o bit de paridade é sempre 0s. O uso dessas paridades é em rede de 9 bits que usam o *mark* e o *space* bits para indicar se os bits de dados contêm um endereço ou dados. O bit de *stop* pode ser configurado como 1, 1.5 ou 2. O bit 1,5 era utilizado em máquinas *teletypewriters*¹ de 60 palavras por minuto, as outras configurações definem quando pinos enviam sinais de “*handshake*”, ou outra checagem de integridade dos dados (AXELSON, 2007).

3.2.1 *Bit Rate e Baud Rate*

A taxa de transferência (*Baud Rate*) é o número de bits por segundo transmitidos ou recebidos por uma unidade de tempo, frequentemente expressa por bits por segundo (bps). O número de caracteres transmitidos por segundo é igual a taxa de bits dividido pelo número de bits na palavra. Com o formato 8-N-1, um byte de dados é transmitido em 1/10 a taxa de bits devido cada palavra conter 10 bits, sendo: 1 *Start* bit, 8 bits de dados e 1 *Stop* bit. Para uma taxa de transmissão de 9600 bps, utilizando uma configuração 8-N-1, pode transmitir 960 bytes a cada segundo (AXELSON, 2007).

3.2.2 O Padrão RS-232

O RS-232 é um padrão definido pela EIA (*Electronics Industries Association*) para comunicação serial bit a bit entre dois dispositivos. O padrão RS-232 define as características mecânica e elétrica. A mecânica trata-se dos conectores macho e fêmea denominados DB9 (D-sub 9²) e a elétrica trata dos níveis de tensão, taxa de sinalização, taxa de rotação dos sinais, nível máximo de tensão, comportamento de curto-circuito e carga máxima da capacitância. Dentre as características elétricas do padrão RS-232, a capacitância da linha de transmissão é limitada em 2500 pF com uma resistência que pode variar entre 3KΩ e 7KΩ, limitando a distância do cabo em 20 metros. O padrão RS-232 oferece suporte para os dois tipos de comunicação assíncrona e síncrona definidas na Seção 3.2, porém, não especifica o protocolo de comunicação entre os dispositivos.

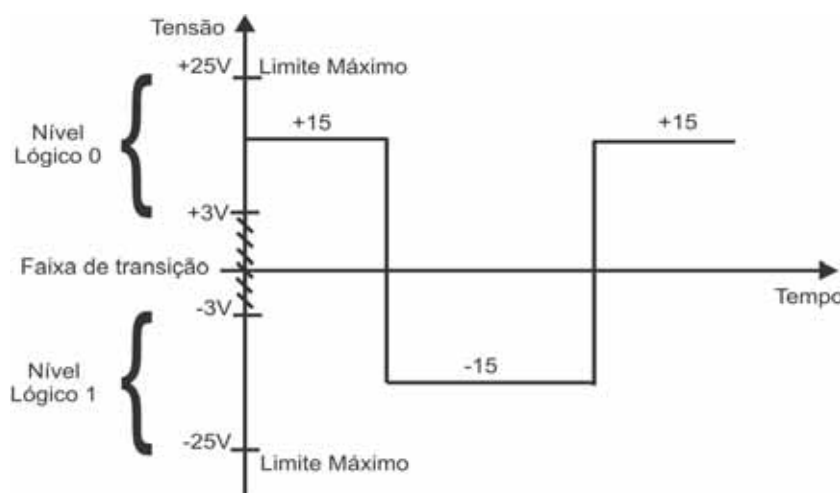
Para realizar a comunicação entre os dispositivos, o padrão RS-232 define como nível

¹ *Teletypewriters* - é uma máquina de escrever eletromecânica para transmissão de dados.

² DB9 - DB representa o encapsulamento dos pinos e 9 a quantidade de pinos utilizado pelo conector.

lógico 1 a faixa de -3V a -15V e o nível lógico 0 a faixa de 3V a 15V. Entre os valores -3V a +3V é chamada de faixa de transição e não possui identificação. Cabe ressaltar que, apesar dos níveis lógicos 0 e 1 sejam reconhecidos em uma faixa de transição -3V e +3V, recomenda-se o transmissor não aplique uma faixa menor que +5V para representar o nível lógico 0 e não aplicar uma tensão menor que -5V para representar o nível lógico 1, sendo que, a margem de 2V é a margem de segurança contra ruído. A margem de segurança de limite máximo para a tensão em cada pino é de +25V e -25V, tendo uma tolerância de 10V, tanto para nível lógico 0 ou 1 (CANZIAN, 2008). Na Figura 14, apresentam-se os níveis de tensão para a representação dos valores lógicos 1 e 0 de acordo com o padrão RS-232.

Figura 14 – Nível de tensão utilizado pelo padrão RS-232.



Fonte: Canzian (2008)

No nó de rede NCAP, o padrão RS-232 foi utilizado para realizar o interfaceamento com STIM, baseando-se, assim, na norma IEEE P1451.2 especificada no Capítulo 2, Seção 2.4.

3.3 *Serial Peripheral Interface Bus - SPI*

O SPI é um padrão de interface serial síncrona nomeada pela Motorola e que opera em modo *full-duplex*. O sistema trabalha em modo mestre/escravo, no qual o mestre inicializa a comunicação dos dados. O sistema permite múltiplos *slaves*, utilizando um pino para seleção através de um único mestre. Para a versão 3 do SPI, denominada SPIV3 (*Serial Peripheral Interface Bus Version 3*), a interface possui as seguintes características: modo mestre e escravo, modo bidirecional, saída de seleção de escravos, controle de operação SPI durante o modo de espera, *clock* serial com programação de fase e polaridade e registrador

de dados buferizado (SEMICONDUCTOR, 2011). O sistema de comunicação SPI utilizado neste trabalho foi de 4 pinos, sendo:

- MOSI (*Master Out/Slave In Pin*) - este pino é definido para saída de dados do módulo SPI quando está configurado como mestre e recebe os dados quando está configurado como escravo;
- MISO (*Master In/Slave Out Pin*) - este pino é definido para saída de dados do módulo SPI quando está configurado como escravo e recebe os dados quando está configurado como mestre;
- \overline{SS} (*Slave Select Pin*) - este pino é usado como sinal de saída do módulo SPI mestre para selecionar o dispositivo que se encontra no mesmo barramento realizando, assim a comunicação com diferentes escravos;
- SCK (*Serial Clock Pin*) - este pino é usado como sinal de saída do clock entre o dispositivo mestre e o escravo.

Na Figura 15, apresenta-se um modelo de mestre/escravo utilizando o barramento SPI com 4 fios para realizar a comunicação.

Figura 15 – Mestre/escravo utilizando barramento SPI.



Fonte: Santos Filho (2012)

A interface SPI foi utilizada no nó de rede NCAP para o interfaceamento com cartão de memória e realizar o armazenamento dos dados relacionados ao NCAP, como por exemplo: a aplicação do NCAP, os TEDS, páginas web e relatório de acesso.

3.4 ZigBee - Padrão IEEE 802.15.4

ZigBee é um padrão que define um conjunto de protocolos de comunicação para redes sem fio de baixa taxa de transferência de dados e curtas distâncias. As redes sem fio baseadas em ZigBee operam na faixa de frequência de 868 MHz, 915 MHz e 2,4 GHz, sendo a taxa máxima de transmissão de 250 Kbps. Uma das principais características do ZigBee é o baixo consumo da bateria fazendo com que o sistema fique em funcionamento

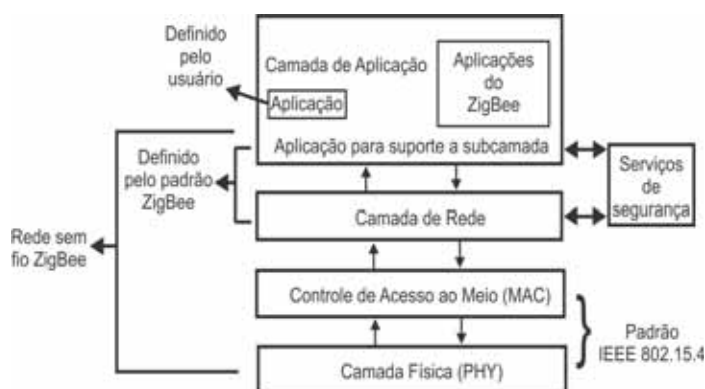
por um longo período de tempo. Tal característica é devido ao dispositivo passar a maior parte do tempo em *sleep mode*, sendo ativado quando for realizar alguma operação. Como resultado, o ZigBee é capaz de operar durante anos sem a necessidade de substituir as baterias (FARAHANI, 2008). O ZigBee pode ser utilizado em diversas aplicações, como: casas inteligentes, agricultura de precisão, indústria e na saúde, realizando, por exemplo, o monitoramento de pressão corporal e temperatura.

O padrão ZigBee foi desenvolvido pela ZigBee Alliance, que tem centenas de empresas associadas, desde a indústria de semicondutores e desenvolvedores de *software* para OEMs - (*Original Equipment Manufacturers*) e instaladores. O padrão ZigBee utiliza a norma IEEE 802.15.4 na sua camada física - PHY e os protocolos de MAC. Devido à utilização do padrão IEEE 802.15.4 como sendo o padrão para camada de enlace do ZigBee, este pode realizar a comunicação com outros diversos dispositivos que utilizam o mesmo padrão IEEE 802.15.4 (FARAHANI, 2008).

3.4.1 Relação entre o ZigBee e o IEEE 802.15.4

Uma maneira de realizar a comunicação entre redes cabeadas e redes sem fios é utilizando o conceito de camadas. Cada camada realiza uma determinada função na rede oferecendo seu serviço à camada superior ou inferior. O ZigBee trabalha com o conceito de camadas baseado no modelo OSI e é representado de acordo com a Figura 16.

Figura 16 – Camadas utilizadas pelo padrão ZigBee baseado no modelo OSI.



Fonte: Farahani (2008)

Na Figura 16, apresentam-se duas camadas inferiores, física e MAC, que são definidas de acordo com o padrão IEEE 802.15.4. O padrão IEEE 802.15.4 foi desenvolvido pelo comitê do padrão IEEE 802 e reconhecido em 2003. É importante salientar que o padrão IEEE 802.15.4 não realiza especificações para camadas superiores como, rede e aplicação implementadas no ZigBee. O ZigBee define as camadas de aplicação e rede sendo que entre

as duas camadas são definidos os serviços de segurança. O IEEE 802.15.4 foi desenvolvido independente do padrão ZigBee, sendo que, é possível desenvolver redes sem fio para curtas distâncias, baseando-se apenas no padrão IEEE 802.15.4 (FARAHANI, 2008).

3.4.2 Tipos de dispositivos

Nas redes sem fio IEEE 802.15.4, existem dois tipos de dispositivos: FFDs (*Full-Function Devices*) e RFDs (*Reduced Function Devices*). Um FFD é capaz de realizar todas as funções descritas pelo padrão IEEE 802.15.4, diferentes dos RFDs, que possuem limites de capacidade. Exemplificando, um FFD pode comunicar com qualquer outro dispositivo da rede, mas um RFD pode comunicar apenas com dispositivos FFD e o processamento e armazenamento de dados dos RFDs são normalmente menores do que os FFDs. Os RFDs são focados para aplicações simples, como, por exemplo, ligar ou desligar uma chave sem que haja a necessidade de um complexo controle do sistema (FARAHANI, 2008).

De acordo com a disponibilidade de funções do dispositivo (FFD ou RFD) e sua posição na rede, os nós podem ser classificados como:

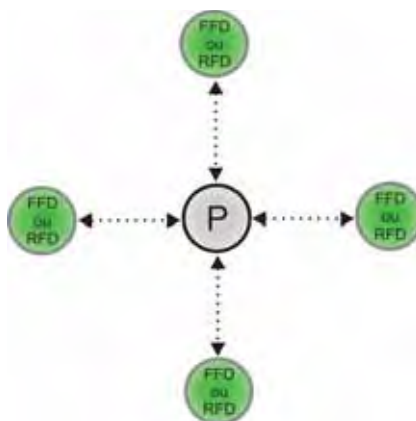
- **Coordenador:** é um dispositivo FFD capaz de realizar transmissões de mensagens. Se o dispositivo é o controlador principal da rede PAN *Personal Area Network*, então é denominado PAN coordenador;
- **Roteador:** é um nó FFD que cria e/ou mantém as informações da rede e a utiliza para determinar a melhor rota para um pacote de dados. O roteador é um dispositivo que pode agir como um coordenador IEEE 802.15.4;
- **Dispositivo final:** é mais barato, possui capacidade de processamento e armazenamento menor, é classificado como um RFD e deve sempre interagir com o seu nó pai (roteador ou coordenador) na rede, para receber ou transmitir dados. O RFDs pode ser também uma fonte ou destino de dados, porém, não possuindo a capacidade de redirecionamento de informações.

3.4.3 Topologia de Rede ZigBee

A formação da rede é gerenciada pela camada de rede ZigBee e é especificada pelo padrão IEEE 802.15.4, que define dois tipos de topologia: estrela e ponta a ponta. Na topologia estrela, todos os dispositivos na rede podem comunicar somente com o coordenador da PAN. Um típico exemplo de formação de rede estrela é quando um FFD

é programado para ser um coordenador. A primeira função do coordenador da rede PAN é definir um identificador único que não é usado por nenhum outro nó da rede na região em que abrange o coordenador. Cada rede possuirá um identificador PAN, diferente dos usados por outras redes que estejam dentro da região de influência das ondas de rádio, permitindo que cada uma das redes opere individualmente. Na Figura 17, apresenta-se um exemplo de topologia estrela, em que a letra “P” representa o dispositivo FFD como coordenador da rede PAN (FARAHANI, 2008).

Figura 17 – Topologia estrela utilizada nas redes ZigBee.

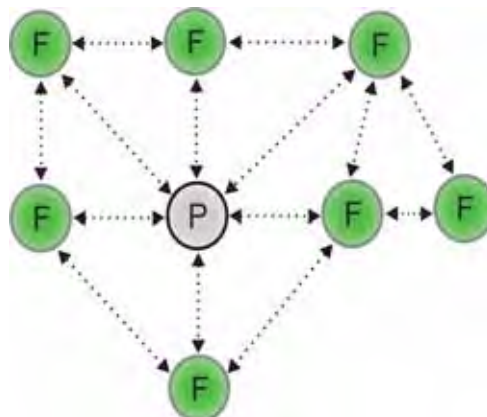


Fonte: Santos Filho (2012)

Na topologia ponta a ponta, cada dispositivo pode comunicar diretamente com qualquer outro dispositivo, se o dispositivo possuir o alcance suficiente para realizar a comunicação. Qualquer FFD em uma rede ponto a ponto pode exercer função de coordenador. Para definir um coordenador na rede é escolhido o primeiro dispositivo FFD que comece a comunicar na rede como coordenador. Em uma rede ponto a ponto, todos os dispositivos que participam da retransmissão são FFDs porque os RFDs não são capazes de retransmitir mensagens. Entretanto, um RFD pode ser parte da rede e comunicar somente com um dispositivo em particular na rede (coordenador ou roteador) (FARAHANI, 2008). Na Figura 18 apresenta-se a topologia ponto a ponto utilizada nas redes sem fio em que a letra “F” representa o FFD.

A interface sem fio ZigBee foi utilizada neste trabalho para realizar a comunicação entre o nó de rede NCAP e o WTIM baseado no padrão IEEE 1451.5 especificado na Seção 2.7. É importante ressaltar que a topologia de rede utilizada neste trabalho foi ponto a ponto, utilizando módulos RFDs.

Figura 18 – Topologia *mesh* utilizada nas redes sem fio ZigBee.



Fonte: Santos Filho (2012)

3.5 O Padrão IEEE 802.3 e a Ethernet

A rede Ethernet é uma tecnologia de LAN com fio que foi desenvolvida em meados da década de 1970. Originalmente, a rede Ethernet foi inventada por Bob Metcalf e Davad Boggs, que teve como projeto inicial um barramento 10Base2³ para interconectar os nós. A rápida discriminação da rede Ethernet foi devido às suas características, sendo: a primeira LAN de alta velocidade, amplamente disseminada e diferente de outras tecnologias FDDI (*Fiber Distributed Data Interface*), *token ring* e ATM (*Asynchronous Transfer Mode*). A tecnologia LAN (*Local Area Network*) é menos complexa e mais barata, o que encorajou ainda mais os administradores na questão de mudanças. Com a Ethernet comutada introduzida no início da década de 90, aumentou ainda mais a velocidade efetiva dos dados, com a Ethernet tornando-se cada vez mais popular, o *hardware* e seus equipamentos (adaptadores, hubs e comutadores) tornaram-se cada vez mais comuns e de baixo custo. Diferente da topologia original da rede Ethernet (barramento), hoje, exceto as instalações herdadas, quase todas as redes Ethernet utilizam topologia estrela de acordo com a Figura 19.

A tecnologia Ethernet foi padronizada através da norma IEEE 802.3 que utiliza o quadro Ethernet para o encapsulamento dos dados e, para a transmissão dos dados utiliza o protocolo de acesso múltiplo da Ethernet, o CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*) (KUROSE; ROSS, 2005). Na Seção 3.5.1 apresenta-se uma descrição do quadro Ethernet e, na Seção 3.5.2 o protocolo CSMA/CD.

³10Base2 - 10Mbps - Baseband - 200 metros

Figura 19 – Topologia estrela utilizada nas redes Ethernet.



Fonte: Santos Filho (2012)

3.5.1 Estrutura do quadro Ethernet

Na rede Ethernet, é utilizado um quadro de dados para realizar o encapsulamento dos protocolos das camadas superiores (Rede, Transporte e Aplicação) e enviar ao meio físico (KUROSE; ROSS, 2005). A estrutura do quadro Ethernet pode ser visualizado na Figura 20.

Figura 20 – Quadro Ethernet.

Preâmbulo (8 bytes)	Endereço de Destino (6 bytes)	Endereço de Origem (6 bytes)	Tipo (2 bytes)	Dados (46 a 1.500 bytes)	CRC (4 bytes)
------------------------	-------------------------------------	------------------------------------	-------------------	-----------------------------	------------------

Fonte: Kurose e Ross (2005)

Os campos apresentados na Figura 20 possuem as seguintes descrições (KUROSE; ROSS, 2005):

- **Campo de dados (46 a 1.500 bytes):** campo destinado ao armazenamento do datagrama IP. O tamanho máximo para realizar o encapsulamento dos dados é de 1.500 bytes, isso significa que, se o datagrama IP exceder o tamanho máximo do campo de dados, o nó cliente terá que fragmentar o datagrama. O tamanho mínimo para o campo de dados é de 46 bytes, sendo que, se este valor for inferior, o campo de dados terá que ser preenchido de modo a completar 46 bytes. Quando o campo utiliza-se da técnica de preenchimento e os dados são passados para a camada de rede, essa camada usa o campo de comprimento do cabeçalho do datagrama IP para remover o dado que foi preenchido;
- **Endereço de destino (6 bytes):** campo destinado ao armazenamento do endereçamento MAC de destino. Quando um adaptador recebe um quadro Ethernet

cujos endereços de destino pertencem ao seu adaptador, o quadro é repassado para a camada de rede. Quando o quadro Ethernet recebido pelo adaptador possui um endereçamento MAC de destino que não corresponde ao próprio adaptador que recebeu, o quadro é descartado;

- **Endereço de origem (6 bytes):** campo destinado ao armazenamento do endereço de origem do adaptador que transmite o quadro para a LAN;
- **Campo Tipo (2 bytes):** o campo tipo permite que a Ethernet multiplexe seus dados, pois, uma máquina cliente pode utilizar vários tipos de protocolos na camada de rede além do protocolo IP, como por exemplo, o protocolo ARP (*Address Resolution Protocol*) que possui seu próprio tipo de dado;
- **Verificação de redundância cíclica - CRC (4 bytes):** campo que permite o destinatário verificar se os dados do quadro foram danificados ou trocados, erros que podem ter ocorrido devido à atenuação da amplitude do sinal ou a energia eletromagnética, ambiente que interfere nos cabos e placas de interface Ethernet;
- **Preâmbulo (8 bytes):** o quadro Ethernet possui um campo que começa com 8 bytes, no qual os 7 primeiros bytes do preâmbulo possuem o valor 10101010_2 e o último byte 10101011_2 . Os 7 primeiros bytes são utilizados para ativar os adaptadores receptores fazendo, assim, o sincronismo dos *clocks* entre ambos, pois, cada adaptador na rede pode possuir diferentes características de desempenho. Os dois primeiros bits consecutivos do oitavo byte alerta o adaptador de que os dados estão chegando e que os próximos 6 bytes são de endereço de destino.

3.5.2 CSMA/CD: *Carrier Sense Multiple Access with Collision Detection*

Uma rede LAN Ethernet interconectada por um hub é conhecida como LAN *Broadcasting*. *Broadcasting* é um termo utilizado quando um adaptador transmite um quadro e todos os adaptadores da rede recebem o quadro. Como emprega o serviço de *Broadcasting*, as redes LAN Ethernet utilizam-se de um protocolo de acesso múltiplo denominado CSMA/CD. As funcionalidades do protocolo podem ser descritas como (KUROSE; ROSS, 2005):

- um adaptador não transmite um quadro quando percebe que algum outro quadro está utilizando o meio de transmissão;

- um adaptador aborta a transmissão quando percebe que algum outro adaptador está transmitindo, ou seja, utiliza detecção de colisão;
- antes de retransmitir um quadro, o adaptador aguarda um tempo aleatório considerável pequeno para reenviar o pacote.

O funcionamento do protocolo CSMA/CD trabalha de forma explícita com outros adaptadores e os passos de execução do protocolo podem ser descritos como (KUROSE; ROSS, 2005):

1. O adaptador recebe o datagrama do quadro Ethernet, prepara e armazena o quadro Ethernet no *buffer* do adaptador;
2. Verifica-se o adaptador está ocioso. Caso esteja, começa a transmitir o quadro, senão, espera perceber que não há energia de sinal e, então, começa a transmitir o quadro;
3. Durante a transmissão dos dados, o adaptador monitora a presença de energia vinda de outros adaptadores. Caso o adaptador transmita o quadro inteiro sem detectar energia de sinal vinda de outros adaptadores, a transmissão será concluída;
4. O adaptador detectando energia de sinal vinda de outros adaptadores enquanto está transmitindo, o adaptador parará a transmissão e enviará um sinal de reforço de 48 bits indicando a colisão;
5. Ao abortar a transmissão, o adaptador espera um tempo aleatório. Ao finalizar o tempo aleatório, retorna à etapa 2.

O padrão IEEE 1451 não especifica a rede externa a ser utilizada no desenvolvimento do nó de rede. Neste contexto, a rede externa utilizada foi a rede Ethernet, uma rede disseminada e utilizada em diversas áreas.

3.6 Considerações Finais Sobre o Capítulo 3

Neste capítulo apresentaram-se as interfaces de comunicação utilizadas para o desenvolvimento do nó de rede embarcado baseado no padrão IEEE 1451. Foram apresentadas suas principais características e onde se encaixam dentro do trabalho, quais as interfaces utilizadas pela norma e quais foram utilizadas como suporte ao desenvolvimento.

Capítulo 4

Ferramentas Utilizadas no Desenvolvimento do Nó de Rede NCAP e dos TIMs

4.1 Introdução

Neste capítulo aborda-se as ferramentas que foram utilizadas no desenvolvimento do nó de rede (IEEE 1451). Na Seção 4.2 trata-se do sistema operacional embarcado μ Clinux, suas características e exemplos de aplicações. Na sequência, serão abordados conceitos específicos relacionados com a tecnologia de lógica programável, abrangendo-se o ambiente, linguagem de descrição de *hardware*, ambiente SoPC Builder, o processador Nios II e os dispositivos lógicos programáveis. Nas Seções 4.7, 4.8, 4.9, 4.10 e 4.11, serão abordados conceitos do servidor Boa, CGI, o pacote Busy Box, o microcontrolador ATmega8 utilizado no desenvolvimento dos TIMs e a linguagem C.

4.2 Sistema Operacional Embarcado μ Clinux

O projeto μ Clinux ou microcontrolador Linux foi iniciada em 1997 por Jeff Dionne, Kenneth Albanowski e outros desenvolvedores, com a meta de criar a versão derivada do Linux Kernel 2.0 para microcontroladores de baixo custo. A primeira versão do μ Clinux foi desenvolvida para um microprocessador Motorola 68000, no qual foi baseado em MC68328 *Dragon Ball Integrated Microprocessor*, com o objetivo de distribuir um sistema de controle SCADA em 1997/1998. O primeiro relato para o público da comunidade de código aberto foi realizado como uma alternativa de sistema operacional para *Palm Pilot* em fevereiro de 1998. Depois desse começo, Jeff Dionne e Michael Durrant projetaram e construíram uma linha para controladores embarcados como uCsim e uCdim. Ao mesmo tempo,

Gerg Ungerer, da mesma empresa μ Clinux, levando para a plataforma Motorola ColdFire, desenvolveu vários sistemas usando esta plataforma base. Neste contexto, o próximo foco de desenvolvimento era criar ambientes do μ Clinux para outras plataformas de *hardware* (NIKKANEN, 2003).

O interesse para esses pequenos processadores foi propagado rapidamente e conduziu para o desenvolvimento de outros *softwares*. Um desses *softwares* foi o uC-libc, biblioteca na qual foi designada a substituir as bibliotecas libe e glibc dentro de um pequeno pacote. Originalmente, o desenvolvimento do μ Clinux foi baseado na versão do Linux Kernel 2.0.33. Na versão do Kernel 2.2, teve somente pequenas mudanças efetivas para o dispositivo sem MMU (*Memory Management Units*). Depois do ano 2000, o Linux 2.4 foi liberado e as mudanças feitas foram suficientemente maiores para estas plataformas sem MMU (NIKKANEN, 2003).

O μ Clinux destina-se a microcontroladores sem MMUs, entretanto, o Linux/*Microcontroller Project* tem crescido tanto em marca, quanto a relação da cobertura entre diversas arquiteturas de processadores. Hoje, o sistema operacional μ Clinux inclui um kernel de 2.0, 2.4 e 2.6 e um conjunto de aplicações para o usuário. O μ Clinux possui diversos programas já conhecidos pelos usuários do Linux, tais como Shell, PPP (*Point-to-Point Protocol*), HTTP e Telnet (DIONNE; DURRANT, 2002).

- **Shell** – é um módulo que atua como interface para o usuário disponibilizando diversas tarefas através de comandos internos do sistema operacional;
- **PPP** – protocolo desenvolvido para permitir acesso autenticado e a transmissão de pacotes de diversos protocolos;
- **HTTP** – protocolo de transferência de páginas web;
- **Telnet** – protocolo utilizado para realizar acesso remoto entre computadores.

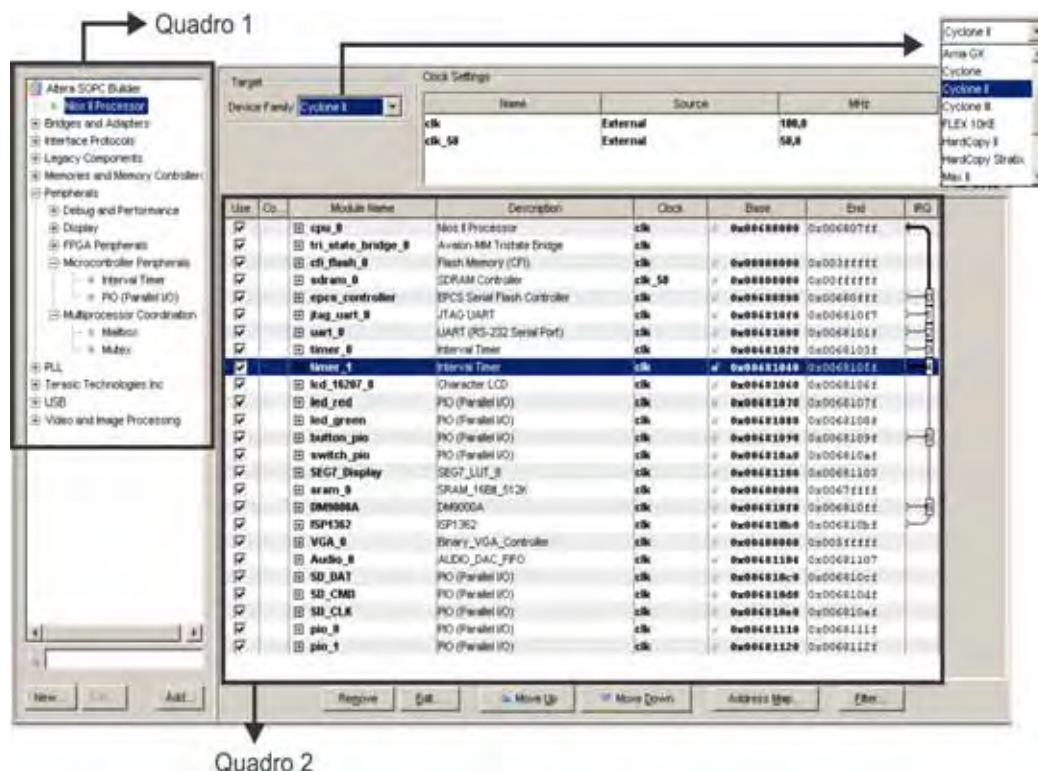
O sistema operacional μ Clinux oferece suporte a diferentes processadores como Axix, Etrax, ARM, Atari 68K, ColdFire, Nios e para muitos outros processadores. Apesar do μ Clinux trabalhar com plataforma de *hardware* e processadores diferenciados, o μ Clinux busca manter as características do Linux convencional. Características como suporte à pilha de protocolo TCP/IP (*Transmission Control Protocol/Internet Protocol*), suporte a sistema de arquivo, algumas aplicações do próprio sistema operacional Linux como FTP e servidor Boa, estando limitado apenas pela característica de *hardware* (JUNQUEIRA, 2007).

4.3 Ambiente SoPC Builder

O SoPC Builder é uma ferramenta integrada com o Quartus II da Altera que permite ao usuário criar sistemas baseados no processador Nios II por simples seleção de blocos. O SoPC possui blocos lógicos de *hardware* pré-definidos, semelhantes ao de um microcomputador, como processador Nios II, controladores de memória, interfaces e periféricos. Ao definir o conjunto de blocos lógicos, o SoPC integra os blocos em um único sistema no qual pode ser visualizado e programado no DE2 (*Development and Education 2*) utilizando o Quartus II (ALTERA, 2006).

Na Figura 21 ilustra-se a interface do ambiente SoPC Builder. Na Figura 21 no Quadro 1, componentes pré-definidos pelo ambiente, como por exemplo, o processador Nios II, memória SDRAM (*Synchronous Dynamic Random Access Memory*), periféricos de I/O e interfaces de comunicação, e, no Quadro 2 indicam-se os componentes que foram definidos dentro do projeto. A opção *Device Family* na Figura 21, representa o FPGA que será utilizado e, na *Clock Settings*, define-se a frequência do *clock* do sistema.

Figura 21 – Interface do ambiente SoPC Builder.



Fonte: Santos Filho (2012)

4.4 Verilog HDL

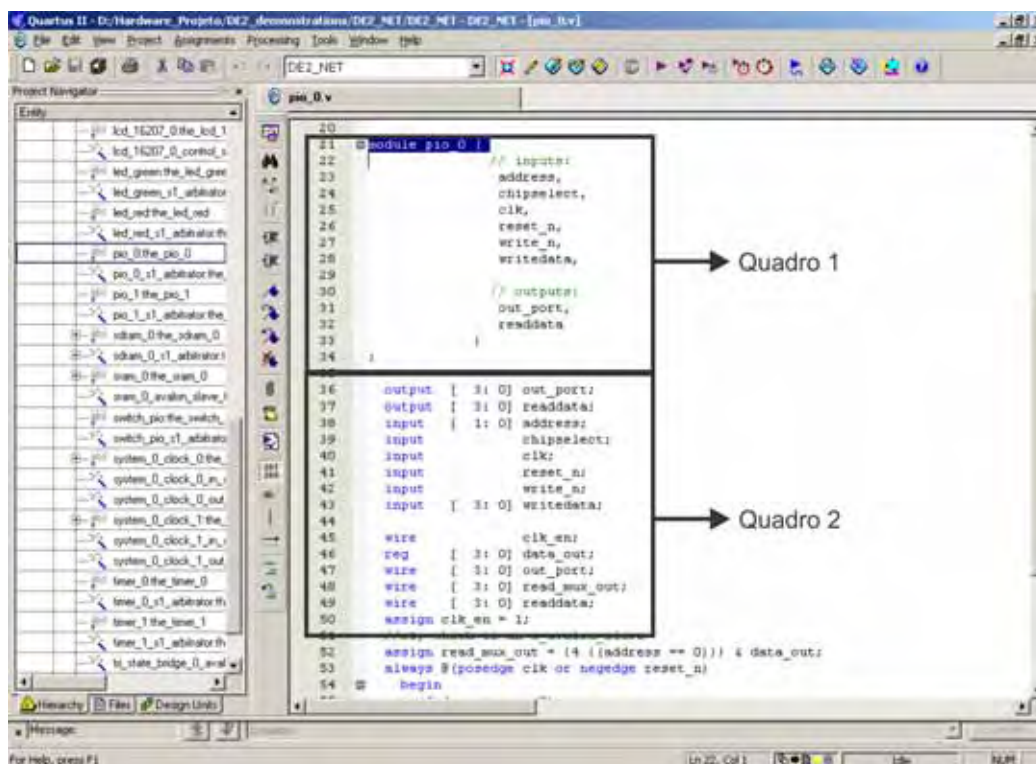
A Verilog HDL (*Hardware Description Language*) é uma linguagem para síntese de *hardware* baseada na linguagem HILO (desenvolvida na Universidade de Brunel na Inglaterra) e iniciado em 1981 por Phil Moorby da Gateway Design Automation. Apesar de criado em 1985, o Verilog ganhou forças a partir de 1987 entre os projetistas de *hardware*, tanto para a descrição quanto para a síntese de sistemas digitais complexos. A primeira ferramenta de síntese lógica para o Verilog foi a Synopsys em 1988. Com o lançamento da ferramenta, o Verilog ganhou um grande impulso devido à ferramenta permitir desenvolvimento de projetos em RTL (*Register Transfer Level*) e fazer tradução para o nível de gate (TECHNOLOGY, 2011).

No ano de 1989, a empresa Cadence comprou todos os direitos do Verilog da Gateway e continuou o projeto tanto com a linguagem quanto com as ferramentas, juntamente com a Synopsys, que estava avançando com suas ferramentas de síntese. Devido à linguagem Verilog ser patenteada pela Cadence, começou a perder espaço no mercado para as linguagens de domínio público, como a VHDL (*VHSIC Hardware Description Language*). Isso fez com que a Cadence tornasse a linguagem Verilog de domínio público (TECHNOLOGY, 2011) .

No ano de 1990, uma comissão denominada OVI (*Open Verilog International*) assumiu a responsabilidade de tornar todos os componentes do Verilog em domínio público. Com isso, a comissão OVI aperfeiçoou e estendeu a linguagem para o uso de outras tecnologias, refinou a documentação do Verilog LRM (*Language Reference Manual*) além de formar o grupo OVI LRM. Com essas modificações, realizou-se a solicitação à IEEE, para a padronização, que posteriormente, foi aceita e definida como IEEE std 1364-1995 em 1995 (TECHNOLOGY, 2011) . Atualmente, a linguagem Verilog possui diversos simuladores para uma grande variedade de sistemas operacionais e cada um de seus simuladores com suas características e vantagens.

Na Figura 22 ilustra-se um exemplo de código do bloco `pio.v` em Verilog, gerado pela ferramenta SoPC Builder. O bloco representa a saída de 4 bits pelos pinos de I/O do DE2. Na Figura 22 no Quadro 1 apresenta-se o *module*, que é a unidade básica e única do Verilog, na qual está contida toda a configuração I/O e onde se insere a função lógica. Na Figura 22 no Quadro 2 apresenta-se o “*Port Declaration*” que é uma subunidade do *module*, no qual possui as configurações de I/O.

Figura 22 – Exemplo de código em Verilog no ambiente Quartus II.



Fonte: Santos Filho (2012)

4.5 Processador Nios II

O Nios II é atualmente a CPU (*Central Processing Unit*) Softcore que apresenta a maior flexibilidade e desempenho entre as opções disponíveis no mercado. O *software* do processador Nios II é configurável e designado a implementações em FPGAs da Altera. Os seguintes processadores Nios II são disponibilizados (ALTERA, 2009):

- **Nios II/f** - o Nios II/f (*fast*) é um processador designado para tarefas de alto desempenho e apresenta mais opções de configurações e necessita de maior quantidade de elementos lógicos;
- **Nios II/s** - o Nios II/s (*standard*) é um processador para o desenvolvimento de projetos pequenos mantendo as principais características;
- **Nios II/e** - o Nios II/e (*economy*) é um processador designado a processamento de arquivos pequenos.

Todos os modelos de processadores são compilados usando o Quartus II versão 6.1 ou versão superior. Na Figura 23 apresentam-se os modelos de processadores Nios II disponíveis no ambiente SoPC Builder.

Figura 23 – Definição dos processadores no ambiente de desenvolvimento SoPC Builder.



Fonte: Santos Filho (2012)

O Nios II pode ser implementado em diferentes modelos de FPGA (Cyclone, Cyclone II, Cyclone III, Stratix, Stratix II, Stratix III, HardCopy e HardCopy Stratix). Este sistema integrado permite ao projetista desenvolver e configurar um sistema completo com CPUs, DMAs (*Direct Memory Access*) e periféricos em pouco tempo, com alto poder de processamento considerado para dispositivos embarcados (ALTERA, 2009). Na Tabela 8, ilustra-se a frequência máxima em MHz de cada processador em diferentes famílias de FPGA 8.

Tabela 8 – Relação de frequência máxima em MHz entre os processadores Nios II.

Família	Dispositivo usado	Nios II/f	Nios II/s	Nios II/e
Stratix IV	EP4SGX230HF35C2	290	250	340
Stratix III	EP3SL70F484C2	290	230	340
Stratix II	EP2S60F1020C3	220	170	285
Stratix	EP1S80F1020C5	150	130	170
HardCopy® III	HC322FF1152	230	220	210
HardCopy® II	HC230F1020C	200	200	320
HardCopy Stratix	EP1S80F1020C5 HC	150	130	175
Cyclone III	EP3C40F324C6	175	145	210
Cyclone II	EP2C20F484C6	140	110	195
Cyclone	EP1C20F400C6	135	120	175
Arria II® GX	EP2AGX95DF25C6	210	190	320
Arria GX	EP1AGX60CF484C6	140	100	150

Fonte: Altera (2009)

Os resultados apresentados na Tabela 9 foram obtidos utilizando o *software* Quartus II da Altera. Os resultados das MIPS (*Millions of Instructions Per Second*) foram obtidos através da análise de Dhrystone¹ versão 2.1. Para a análise, foram utilizadas as seguintes características de *hardware* como referência (ALTERA, 2009):

- Processador Nios II/f (versão 6.1);
- JTAG (*Joint Test Action Group*) UART (*Universal Asynchronous Receiver/Transmitter*);
- 64 Kbytes memória *on-chip* (Cyclone uso designado a 1 Mbyte de off-chip SDRAM (*Synchronous Dynamic Random Access Memory*)).

Na Tabela 9 é ilustrada a relação de MIPS de acordo com processador Nios II e o FPGA utilizada.

Tabela 9 – Relação das MIPS de acordo com cada processador Nios II.

Família	Dispositivo Usado	Nios II /f	Nios II/s	Nios II /e
Stratix IV	EP4SGX230HF35C2	340	150	48
Stratix III	EP3SL70F484C2	340	140	48
Stratix II	EP2S60F1020C3	250	110	45
Stratix	EP1S80F1020C5	170	80	27
HardCopy® III	HC322FF1152	260	140	30
HardCopy® II	HC230F1020C	230	130	50
HardCopy Stratix	EP1S80F1020C5 HC	165	85	27
Cyclone III	EP3C40F324C6	195	90	30
Cyclone II	EP2C20F484C6	145	55	18
Cyclone	EP1C20F400C6	130	52	17
Arria II® GX	EP2AGX95DF25C6	240	120	50
Arria GX	EP1AGX60CF484C6	150	65	25

Fonte: Altera (2009)

O processador Nios II pode ser aplicado em diversos ambientes, como por exemplo:

- Sistemas que necessitam alta capacidade de processamento e que possam ser reconfigurados independente de sua localização;
- Em sistemas autônomos como roteadores de pacotes de rede Ethernet;
- Sistemas dedicados;

¹Dhrystone - é um programa usado na análise da eficiência de combinações *hardware*/compilador em máquinas de pequeno e médio porte. O original ainda é utilizado para medir desempenho de processador.

- Sistemas onde é necessária mais de uma CPU;
- Sistemas de processamento de imagem;
- Sistemas de processamento de áudio.

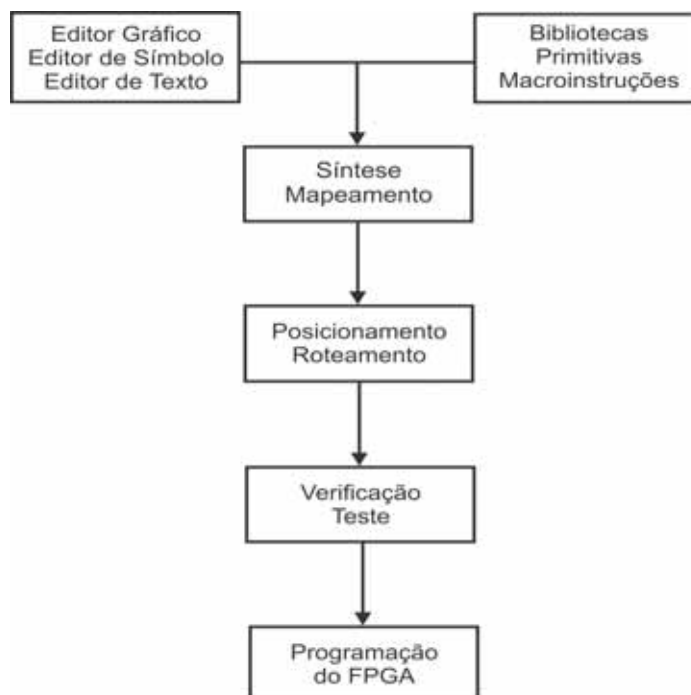
4.6 Dispositivo Lógico Programável e a plataforma de Desenvolvimento DE2

O FPGA consiste em arranjo de células lógicas, ou blocos lógicos configuráveis, contidos em um único circuito integrado. Cada célula contém capacidade computacional para implementar funções lógicas e realizar o roteamento para comunicação entre os blocos. O FPGA basicamente possui blocos lógicos, blocos de entrada e saída e chaves de interconexão. Os blocos lógicos formam uma matriz bidimensional e as chaves de interconexão são organizadas como canais de roteamento horizontal e vertical entre as linhas e colunas de blocos lógicos. Esses canais de roteamento possuem chaves de interligação programáveis que permitem conectar os blocos lógicos de maneira conveniente, em função das necessidades de cada projetista (COSTA, 2005).

O desenvolvimento de projetos com FPGA envolve algumas etapas que, geralmente, são automatizadas e utiliza *software* EDA para simplificar e acelerar o processo. O processo de desenvolvimento com ferramentas EDA envolve as seguintes etapas e que são ilustradas na Figura 24, especificação e entrada do projeto, síntese e mapeamento da tecnologia, posicionamento e roteamento, verificação e teste, programação do FPGA (COSTA, 2005) .

Dentre as famílias de FPGA disponibilizadas comercialmente pela Altera, têm-se: Stratix II, Stratix, StratixGX, Cyclone II, APEX, APEX II, APEX 20K, Mercury, FLEX 10K, ACEX 1K, FLEX 6000 e Excalibur Devices. Os FPGAs também são utilizados em muitos kits didáticos que facilitam o aprendizado e o desenvolvimento de sistemas. Alguns desses kits possuem periféricos semelhantes aos microcomputadores, como por exemplo: a rede Ethernet, saída de vídeo VGA, conector PS/2, interface USB, memória SRAM, SDRAM e memória flash. Tais kits também podem conter apenas periféricos específicos para uma determinada aplicação. Na Figura 25 apresenta-se o kit DE1 e, na Figura 26 apresenta-se o kit DE2 utilizado no desenvolvimento do nó de rede.

Figura 24 – Etapas de desenvolvimento em ambiente EDA.



Fonte: Costa (2005)

Figura 25 – Kit DE1 Altera.



Fonte: Altera (2008)

Na Tabela 10, apresentam-se as características dos kits DE1 e DE2 e seus periféricos.

Figura 26 – Kit DE2 Altera.



Fonte: Altera (2008)

Tabela 10 – Característica de cada kit DE.

Características	DE1	DE2
Altera Cyclone II FPGA	2C20	2C35
Número de Elementos Lógicos	20 000	35 000
Push-button	4	4
Switch	10	18
LEDs	8 verdes 10 vermelhos	9 verdes 18 vermelhos
Display 7 segmentos	4	8
USB Blaster para programação	X	X
8Mbyte SDRAM	X	X
512K byte SRAM	X	X
4Mbyte memória Flash	X	X
SD card	X	X
Transmissão RS-232 e conector com 9 pinos	X	X
Conector PS/2 mouse e teclado	X	X
2 conectores de 40 pinos de expansão	X	X
Áudio Codec com line-in, line-out, mic-in	X	X
Conector de fêmea VGA	X	X
Modulo LCD 16 x 2	—————	X
Decodificador de TV e conector de entrada	—————	X
Controlador Ethernet 10/100	—————	X
Controlador USB máquina/escravo	—————	X
IrDA (<i>Infrared Data Association</i>)	—————	X

Fonte: Altera (2008)

4.7 O Servidor Boa

O Boa é um servidor web com um padrão diferente dos outros servidores, pois trabalha com apenas um processo. Isto significa que o *fork()*² não é executado para

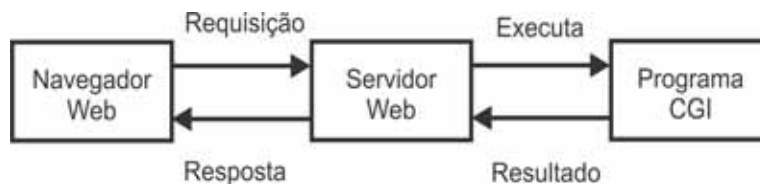
²*fork()* - cria um processo filho que se diferencia do processo pai nos números de PID (*Process Identification*). É importante destacar que, o novo processo não herda a lista dos recursos associados

cada conexão. Em vez disto, o *fork()* é executado apenas para o programa CGI (*Common Gateway Interface*) e, automaticamente, realizado o gerenciamento do diretório com alta performance e segurança. O Boa é desenvolvido diretamente em plataforma Linux para uma variedade de distribuições, como Red HAT Linux, Debian GNU/Linux e o Gentoo Linux. O tamanho do servidor Boa é de 61 KBytes para a distribuição μ Clinux 2.4.17. Além de seu tamanho, o Boa possui também segurança, velocidade e código aberto (WENTWORTH, 2001).

4.8 *Common Gateway Interface - CGI*

O CGI é um protocolo que descreve a conexão entre servidores web e os programas executáveis externos. Determinada característica provê uma diferença nos conteúdos dos arquivos HTML (*HyperText Markup Language*) de acordo com o ponto de requisição. As páginas podem ser realizadas utilizando programação em C/C++, Perl ou programação Shell. A função básica do CGI é apresentada na Figura 27. Ao realizar uma requisição através de uma página web, o servidor realiza a requisição ao CGI, que obtêm as variáveis passadas pelo servidor e executa o programa retornando para o servidor o resultado. O servidor, ao receber o resultado, envia para o navegador web no qual é apresentado para o cliente (GUNDAVARAM, 1996).

Figura 27 – Mecanismo de execução do CGI.



Fonte: Santos Filho (2012)

4.9 BusyBox

O BusyBox é uma simples implementação executável de muitos utilitários padrões do Linux, contém, utilitários como *cat* e *echo* e ferramentas maiores e mais complexas como *grep*, *find*, *mount* e Telnet. A vantagem do BusyBox é compartilhar características comuns entre aplicativos como, por exemplo, *grep* e *find*. Quando os utilitários são combinados dentro de um simples pacote, tais utilitários podem compartilhar elementos comuns em um único executável, resultando em um executável menor. O BusyBox pode ao processo pai, tais como arquivos abertos e sinais pendentes.

empacotar quase 3,5 MB de utilitários dentro, em torno de 200 KBytes. Isto provê uma ótima funcionalidade para discos de inicialização e dispositivos embarcados Linux (JONES, 2006).

4.10 Os Microcontroladores

Os MCUs (*Micro Controller Unit*) surgiram em meados da década de 80 e, ao contrário dos microprocessadores, são dispositivos mais simples, com memória RAM (*Random Access Memory*) e ROM (*Read Only Memory*) internas, oscilador interno de *clock*, I/O interno entre outros. Tais características tornam simples o projeto de dispositivos inteligentes, pois as MCUs raramente precisam de CIs (Circuito Integrado) externos para funcionar, o que contribui para o seu baixo custo e tamanho. Os microcontroladores foram projetados para comandarem equipamentos específicos. Estes equipamentos são conhecidos como *Embedded System*, pois o microcontrolador é embutido em um sistema fechado e com funções bem específicas. Muitas vezes, esses sistemas exigem um controle preciso do tempo, pois trabalham em tempo real e podem executar tanto tarefas simples, como o controle de um portão de garagem ou até mesmo tarefas mais complexa como o controle de um equipamento industrial (PEREIRA, 2007).

Neste trabalho, foram utilizados microcontroladores da linha Atmel, o ATmega8 para o desenvolvimento dos TIMs. O ATmega8 é um microcontrolador CMOS (*Complementary Metal-Oxide-Semiconductor*) de 8 bits, de baixo consumo de energia baseado na arquitetura AVR RISC que executa 1 MIPS por MHz. O ATmega8 possui as seguintes características: 8 KBytes de memória Flash, 512 Bytes de memória EEPROM, 1 KBytes de SRAM interna, 3 canais de PWM (*Pulse-Width Modulation*), 6 canais de ADC (*Analog-to-Digital Converter*) com precisão de 10 bits, interface UART, 23 pinos de I/O, etc (ATMEL, 2011) .

4.11 Linguagem C

A linguagem C teve sua origem no começo dos anos 70, iniciada por Dennis Ritchie. A linguagem C é uma derivação de outra linguagem, conhecida por BCPL (*Basic Combined Programming Language*), desenvolvida por Martin Richards, que mais tarde deu origem à linguagem B, desenvolvida por Ken Thompson, a qual levou ao desenvolvimento da linguagem C (SCHILDT, 1997) .

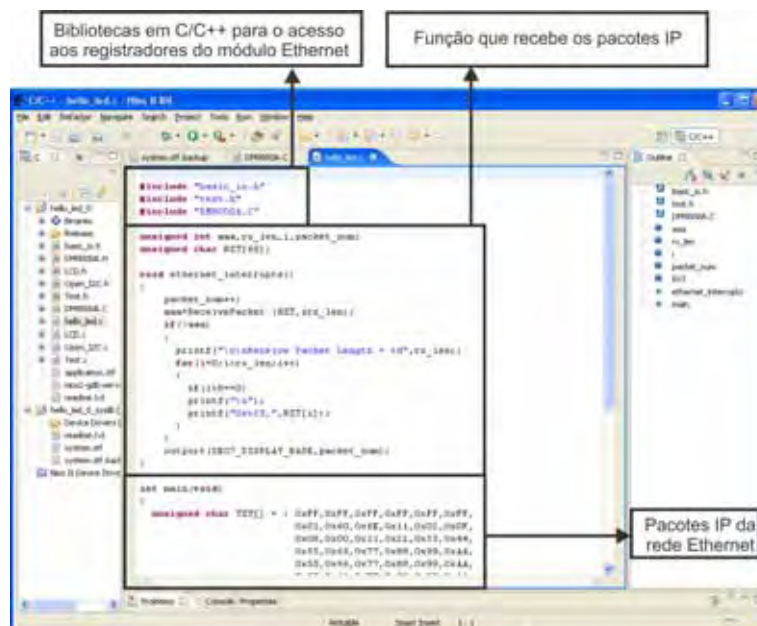
A linguagem C é uma das linguagens mais usadas, sendo flexível e robusta, porém,

tais características não simplificavam a complexidade de entendimento do *software* com grandes quantidades de linhas. Para resolver esse problema, acrescentaram-se várias extensões à linguagem C, dando nome de “C com Classes” que, futuramente, foi denominada C++ (SCHILDT, 1997).

Hoje, a linguagem C/C++ é utilizada tanto para alto nível quanto para baixo nível em diversos ambientes como: microcontroladores, desenvolvimento de sistemas operacionais, programação em dispositivos embarcados, em microprocessadores e kits de desenvolvimento. Deste modo, a linguagem C/C++ é uma linguagem atraente para o desenvolvimento de *softwares* em baixo nível (SCHILDT, 1997).

Na Figura 28 apresenta-se a interface de desenvolvimento Nios II IDE que utiliza para acessar os periféricos a linguagem C/C++. O código da Figura 28 é um *software* para recepção e transmissão de pacotes através da rede Ethernet, utilizando o circuito integrado DM900A disponibilizado pelo Kit DE2 da Altera. O código possui bibliotecas pré-definidas pelo fabricante como a DM9000A.c que, também, é desenvolvida em linguagem C e que possui funções do tipo *ReceivePacket* para receber os pacotes enviados por outros dispositivos.

Figura 28 – Ambiente de desenvolvimento Nios II.



Fonte: Santos Filho (2012)

Outro exemplo de acesso ao *hardware*, utilizando a linguagem C, pode ser visualizado na Figura 29. Em que o código é executado no sistema operacional μ Clinux. Na Figura 29 na linha 1, refere-se à chamada da biblioteca *stdio.h*, as linhas 2 e 3 são as definições das funções *IORD* (leitura de dados) e *IOWR* (escrita dos dados). Para as funções *IORD*

e IOWR, os dados são escritos ou realiza a leitura diretamente do barramento Avalon, de acordo com o endereçamento de cada componente definido no ambiente SoPC Builder. No exemplo do código da Figura 29 é realizada a leitura dos pinos do DE2 através do endereço 0x006810a0 apresentado na linha 11 e a escrita do valor apresentado na linha 15 nos *displays 7 segment* utilizando o endereço 0x00681103. O segundo parâmetro 0 (zero) na função IORD representa o *off-set*, e o terceiro parâmetro na função IOWR representa o valor inteiro que será escrito no barramento e apresentado nos *Displays*.

Figura 29 – Exemplo de código em C para acesso aos periféricos no ambiente μ Clinux.

```

1 - #include <stdio.h>
2 - #define IORD(address,offset) (*(volatile unsigned *)(((address)0x80000000)+4*(offset)))
3 - #define IOWR(address,offset,value) (*(volatile unsigned *)(((address)0x80000000)+4*(offset)))=(value)
4 - int main(void)
5 - {
6 -     int a=0,b;
7 -     a=IORD(0x006810a0,0);
8 -     while(1)
9 -     {
10 -         printf("Leitura do Sensor Strain Gage: \n");
11 -         a=IORD(0x006810a0,0);
12 -         b=IOWR(0x00681120,0,0);
13 -         sleep(1);
14 -         printf("Escrevendo nos Displays o valor %i \n",b);
15 -         IOWR(0x00681103,0,b);
16 -         sleep(1);
17 -     }
18 - }
~

```

Fonte: Santos Filho (2012)

4.12 Considerações Finais Sobre o Capítulo 4

Neste capítulo foram apresentadas as tecnologias utilizadas para o desenvolvimento do projeto como, o sistema operacional μ Clinux, kit DE2 Altera, linguagem de programação C e alguns componentes que serão utilizados durante o desenvolvimento do projeto como, a ferramenta BusyBox.

Capítulo 5

Descrição dos TIMs Desenvolvidos para Testes com o Nó de Rede Embarcado NCAP

5.1 Introdução

Neste capítulo, apresentam-se as características dos TIMs e os módulos desenvolvidos para realização dos testes com o nó de rede embarcado. Neste trabalho foram desenvolvidos dois modelos de TIMs, o primeiro, utilizando interface cabeada (padrão RS-232) que é apresentada na Seção 5.3 e, o segundo modelo, a interface sem fio (padrão IEEE 802.15.4), apresentada na Seção 5.4.

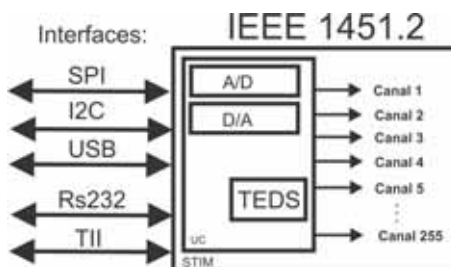
5.2 Descrição Geral de Desenvolvimento dos TIMs - IEEE 1451

O padrão IEEE 1451 introduz o conceito de TIM. No contexto do padrão, os transdutores consideram-se como parte do TIM. Desta forma, para fornecer a capacidade de *plug and play* de cada transdutor, deve-se permanecer inseparável do TIM durante seu funcionamento normal. Neste contexto, cada transdutor que faz parte do TIM é chamado de canal de transdutor e cada canal é considerado inteligente devido a 3 fatores:

- É descrito através das especificações armazenadas em formato eletrônico em um dispositivo de memória não volátil;
- O controle e os dados associados são digitais;
- São fornecidas funções de controle, estado e disparo para suportar a funcionalidade própria de cada canal.

Um TIM pode ser composto de 1 até 255 transdutores por módulo e cada módulo pode conter circuito de condicionamento de sinal, conversores A/D e D/A, lógica necessária para realizar a comunicação através de uma interface padronizada e uma memória para o armazenamento dos TEDS. Cabe ressaltar que as interfaces entre o transdutor e os circuitos de condicionamento de sinal ou entre o circuito de condicionamento e o conversor são realizadas de acordo com critérios estabelecidos de acordo com cada projetista, tornando-se obrigatoriedade a descrição dos TEDS de acordo com o modelo desenvolvido. Na Figura 30, apresenta-se um exemplo de TIM, com saída para interface padronizada (IEEE 1451.X), alguns dispositivos que podem ser utilizados para realizar a aquisição dos dados ou controle do sistema e uma memória não volátil para o armazenamento dos TEDS.

Figura 30 – Exemplo de TIM baseado no padrão IEEE P1451.2.



Fonte: Santos Filho (2012)

O TIM é controlado pelo NCAP através de uma de suas interfaces padronizadas. Deste modo, quando conectado a uma rede utilizando o NCAP, o conjunto é reconhecido pela rede como um ponto de inteligência distribuído. Tal característica possibilita o funcionamento de transdutores em sistemas de medição e controle distribuído.

A funcionalidade do TIM pode ser descrito através de três estados de operação: inicialização do TIM, TIM ativo e TIM inativo. O TIM é colocado no estado de inicialização por um comando de reinicialização ou por um evento para ligar o sistema. Uma vez completado o processo de inicialização do sistema, ocorrerá uma transição passando para o estado TIM ativo como apresenta-se na Figura 31. Após realizar a inicialização, o TIM passará para o estado TIM ativo aguardando o próximo comando. Quando o TIM está no estado TIM ativo, o TIM entrará no estado TIM inativo através de um comando enviado pelo usuário, denominado na Figura 31 como Comando inativo. Para mudar o estado do TIM de inativo para ativo, pode ser enviado um comando de ativar (Comando ativar), configuração personalizada (Personalizado) ou quando finalizar o tempo pré-determinado do estado TIM inativo definido pelo projetista (Fim de tempo) (IEEE, 2007c). Na Figura 31 apresenta-se os estados de operação do TIM e todas suas transições.

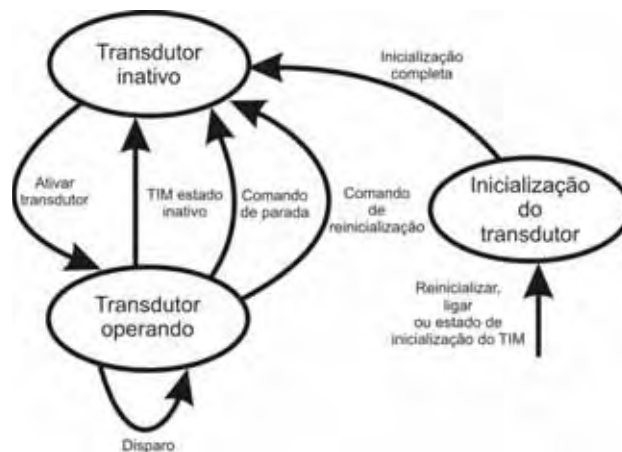
Figura 31 – Diagrama de estado de operação do TIM.



Fonte: IEEE (2007c)

Os estados de operação do canal de transdutor é apresentado na Figura 32. Existem dois tipos básicos de estados de operação do canal de transdutor que podem ser realizados após ser inicializado, denominados de: estado transdutor inativo e transdutor operando. O transdutor entra em estado de operação através de um comando de ativação e permanece neste estado até que seja recebido um comando de parada ou de inicialização; ou o TIM é reinicializado.

Figura 32 – Diagrama de estado de operação do TransducerChannel.



Fonte: IEEE (2007c)

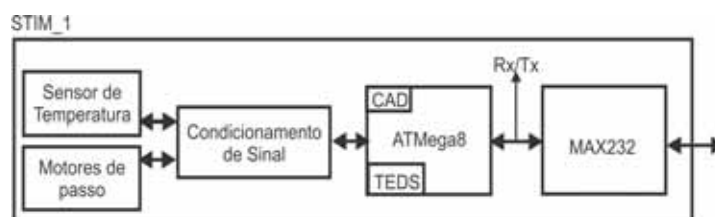
Com base nas características descritas pelo padrão IEEE 1451, neste trabalho foram desenvolvidos 4 TIMs: 2 STIMs e 2 WTIMs que serão descritos nas Seções 5.3 e 5.4.

5.3 *Smart Transducer Interface Module - STIM*

Os STIMs desenvolvidos neste trabalho consistem de uma interface de comunicação RS-232 com o nó de rede NCAP e um circuito integrado MAX232 em cada módulo. A comunicação entre o microcontrolador e o circuito integrado MAX232 foi realizada através de uma interface serial Rx/Tx, utilizando como padrão: 4800 bps - velocidade, 8 - bits de dados, N - sem bits de paridade e 1 - *Stop bit*. Os TEDS do STIMs foram armazenados na memória interna do próprio microcontrolador ATmega8 e podem ser visualizadas os campos descritos e as tabelas no Apêndice C na Seção C.1. Para testes dos STIMs, foram implementados sensores e atuadores, como por exemplo: sensor de temperatura LM35, motor de passo e ventiladores (*coolers*). No Apêndice D apresenta-se a descrição dos transdutores utilizado para o desenvolvimento do módulo STIM.

Os STIMs, desenvolvidos foram denominados de STIM_1 e STIM_2. O STIM_1 foi desenvolvido utilizando 2 motores de passo e um sensor de temperatura e, no STIM_2, foram utilizados 3 ventiladores. Para o STIM_1, o *driver* de controle dos motores de passo foram realizados através de programação no microcontrolador ATmega8, utilizando 4 bits de saída para cada motor de passo e um circuito integrado (ULN2803). Para aquisição da temperatura, foi utilizado o amplificador operacional LM324N e um conversor A/D interno do microcontrolador ATmega8. Na Figura 33 apresenta-se o diagrama de bloco do STIM_1 e, no Apêndice E.1, apresenta-se o diagrama do circuito.

Figura 33 – Diagrama de bloco do STIM_1.

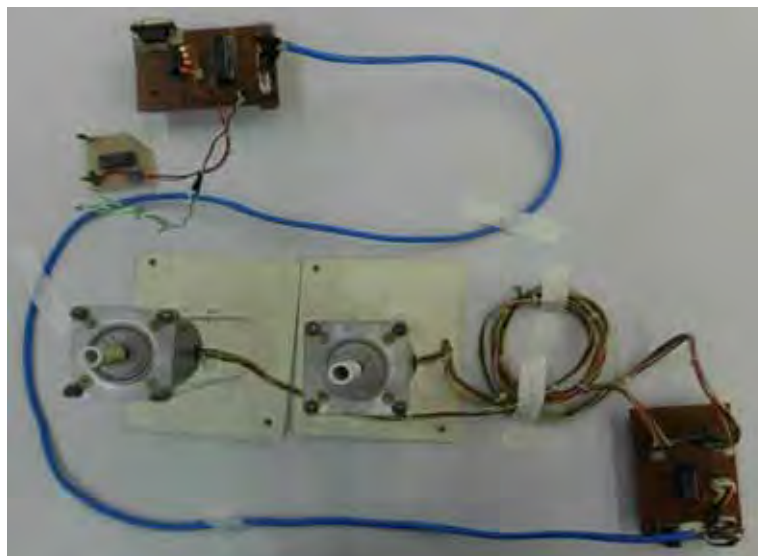


Fonte: Santos Filho (2012)

O STIM_1 ao ser inicializado entra em estado TIM Ativo aguardando um comando do módulo NCAP. Ao receber o comando, realiza a verificação referente ao tamanho do pacote e à integridade dos dados. Após a verificação, o STIM_1 executa o comando com base na classe de comando, função do comando e o canal do transdutor, de acordo com o padrão IEEE 1451.0. Para ativar o motor de passo, o comando enviado para o módulo STIM_1 possui a quantidade de passos o motor realizará e a direção (horário/ anti-horário). Ao finalizar a sequência, o motor para em um determinado valor especificado pelo comando e aguarda o próxima comando. Ao receber um novo comando, o motor de passo continua a executar a tarefa a partir do último valor de sequência. Para o sensor de temperatura o

STIM recebe o comando e realiza a leitura de acordo com o canal do transdutor e retorna o valor através de um comando de resposta. No Apêndice B na Seção B.1 apresentam-se os comandos para o controle do motor de passo e a leitura do sensor de temperatura. Na Figura 34 apresenta-se o STIM_1 desenvolvido em laboratório com dois motores de passo e um sensor de temperatura.

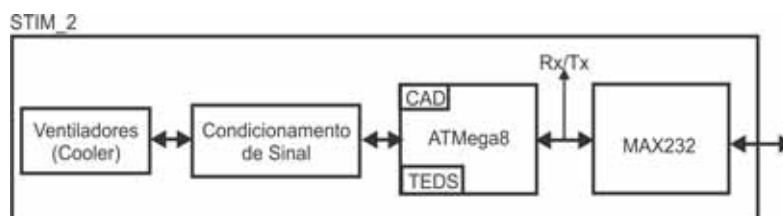
Figura 34 – STIM_1 desenvolvido em laboratório.



Fonte: Santos Filho (2012)

O STIM_2 foi desenvolvido utilizando 3 pinos de I/O do microcontrolador ATmega8 que foi interligado no circuito integrado ULN2803 para acionamento dos 3 ventiladores. Os TEDS do módulo STIM_2 foram armazenados na memória interna do próprio microcontrolador ATmega8 e podem ser visualizados os campos descritos e as tabelas no Apêndice C na Seção C.2. Na Figura 35 apresenta-se o diagrama de bloco do STIM_2 e no Apêndice E.2 o diagrama do circuito desenvolvido.

Figura 35 – Diagrama de bloco do STIM_2.



Fonte: Santos Filho (2012)

O STIM_2, ao ser inicializado, entra em estado Ativo e aguarda o comando do nó de rede NCAP. Ao receber o comando, o STIM_2 realiza a verificação da integridade do comando se não houve perda dos dados durante a transmissão. Após finalizar a verificação

dos dados, o STIM_2 verifica a classe de comando, a função do comando e o canal do transdutor, em seguida, executa o comando. Para o acionamento dos ventiladores, foram definidos os valores 00_H e 01_H , sendo, o valor 00_H desligado e 01_H ligado. No Apêndice B na Seção B.2, apresentam-se os comandos para o controle dos ventiladores. Na Figura 36, apresenta-se o STIM_2 desenvolvido em laboratório com 3 ventiladores.

Figura 36 – STIM_2 desenvolvido em laboratório.



Fonte: Santos Filho (2012)

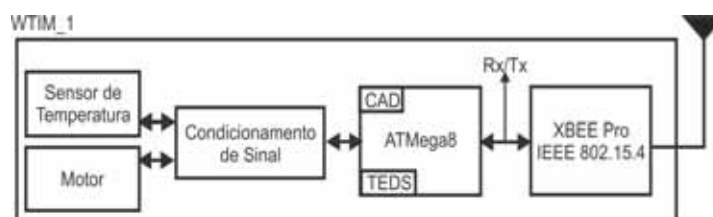
5.4 *Wireless Transducer Interface Module - WTIM*

Os WTIMs denominados WTIM_1 e WTIM_2 foram desenvolvidos utilizando microcontrolador ATmega8 e módulos XBee-Pro. Cada WTIM possui uma fonte de alimentação de 5V e um regulador de tensão LM1117 com faixa de entrada de 5V a 12V e saída de 3.3V, que foi usada para alimentação do módulo XBee-Pro. A comunicação entre o microcontrolador e o módulo XBee-Pro foi realizada através de uma interface serial Rx/Tx, utilizando como padrão: 4800 bps - velocidade, 8 - bits de dados, N - sem bits de paridade e 1 - *Stop bit*. Os TEDS do módulo WTIM_1 foram armazenadas na memória interna do próprio microcontrolador ATmega8 e pode ser visualizado os campos descritos e as tabelas no Apêndice C na Seção C.3.

Para a configuração dos módulos XBee-Pro, foi utilizado o ambiente X-CTU da Digi-MaxStream e um adaptador USB-Bee que realiza a comunicação do módulo XBee-Pro com o microcomputador através de uma interface USB. No Apêndice F apresentam-se os módulos, configurações e os comandos para a configuração dos módulos XBee-Pro através de linha de comando.

Os WTIMs desenvolvidos trabalham de forma ponto a ponto e realiza a comunicação diretamente com o coordenador da rede, sendo que, os demais nós não comunicam entre si. Para testes do sistema, o WTIM_1 foi desenvolvido com 1 sensor de temperatura LM35 e 1 motor CC (Corrente Contínua). Para o condicionamento do sinal do sensor de temperatura, foi utilizado um amplificador operacional LM324N, conversor A/D interno e, para o motor CC, foi utilizado um circuito integrado ULN2803. Na Figura 37 apresenta-se o diagrama de bloco do WTIM_1 e, no Apêndice E.3, o diagrama do circuito.

Figura 37 – Diagrama de bloco do WTIM_1.



Fonte: Santos Filho (2012)

Os comandos utilizados para requisição da temperatura e para acionamento do motor CC são apresentados no Apêndice B na Seção B.3. Na Figura 38 apresenta-se o WTIM_1 desenvolvido em laboratório.

Figura 38 – WTIM_1 desenvolvido em laboratório.

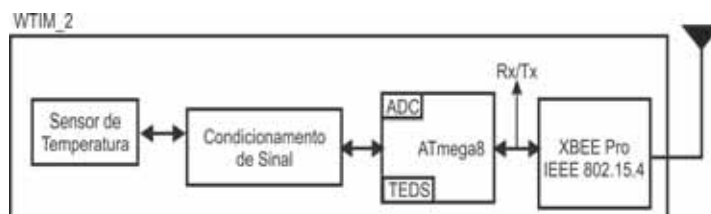


Fonte: Santos Filho (2012)

O WTIM_2 foi desenvolvido utilizando 1 sensor de temperatura, o LM35. Para realizar o condicionamento de sinal do sensor de temperatura, foi utilizado o amplificador operacional LM324N e o conversor A/D interno do microcontrolador. Os TEDS do módulo WTIM_2 foram armazenados na memória interna do próprio microcontrolador ATmega8 e podem ser visualizado os campos descritos e as tabelas no Apêndice C na Seção C.4. Na

Figura 39, apresenta-se o diagrama de bloco do WTIM_2 e, no Apêndice E.4, diagrama do circuito.

Figura 39 – Diagrama de bloco do WTIM_2.



Fonte: Santos Filho (2012)

O comando utilizado para requisição da temperatura é apresentado no Apêndice B, na Seção B.4. Na Figura 40, apresenta-se o WTIM_2 desenvolvido em laboratório.

Figura 40 – WTIM_2 desenvolvido em laboratório.



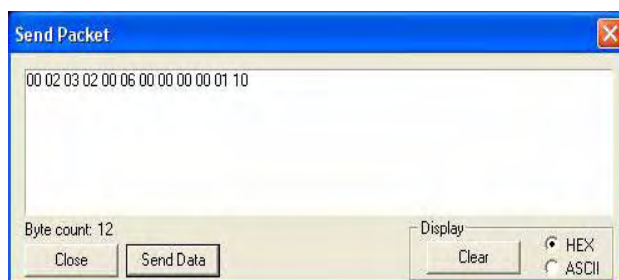
Fonte: Santos Filho (2012)

5.5 Testes realizados com os STIMs e WTIMs

Para testes dos TIMs, foi utilizado o *software* X-CTU, que, através do ambiente é possível enviar protocolo de requisição e disponibilizar a resposta em nível de octeto. Os protocolos foram enviados de acordo com o padrão IEEE 1451.0 - 2007, em que foi especificado de forma sucinta na Seção 2.2. Para os testes dos STIMs, foi utilizada a saída serial RS-232 do microcomputador em que é disponibilizada no ambiente X-CTU como porta “COM1” e a opção “Terminal” do ambiente que apresenta os valores enviados e recebidos em hexadecimal. O primeiro teste foi o controle do motor de passo contido no STIM_1 em que foi enviado o comando de acordo com a Figura 41, descrito no Capítulo 2

na Seção 2.2.3. É importante ressaltar que o penúltimo octeto representa a direção sendo que 0/1 representando horário/anti-horário. O último octeto representa quantidade de passos serão realizados pelo motor de passo.

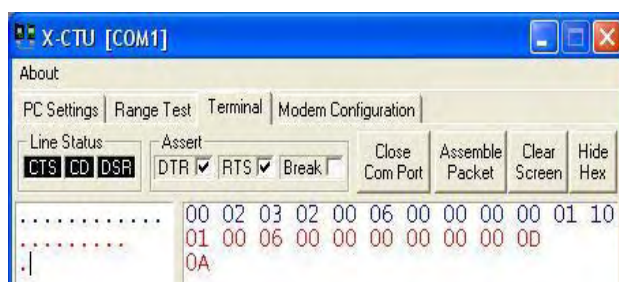
Figura 41 – Protocolo de requisição de controle do motor de passo.



Fonte: Santos Filho (2012)

Como resposta do STIM_1 ao controle do motor de passo, é retornado o valor apresentado na Figura 42, em que a segunda linha representa o protocolo de resposta.

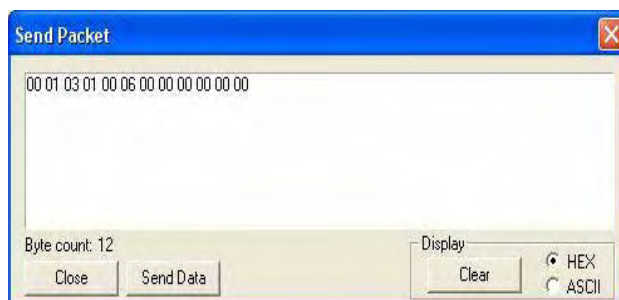
Figura 42 – Protocolo de resposta de controle do motor de passo.



Fonte: Santos Filho (2012)

Para a leitura do sensor de temperatura no WTIM_1, foi utilizado o protocolo de acordo com a Figura 43, descrito no Capítulo 2 na Seção 2.2.3, no entanto, foi utilizada a porta "COM3" para a comunicação com o módulo USB-Bee.

Figura 43 – Protocolo de requisição da temperatura.

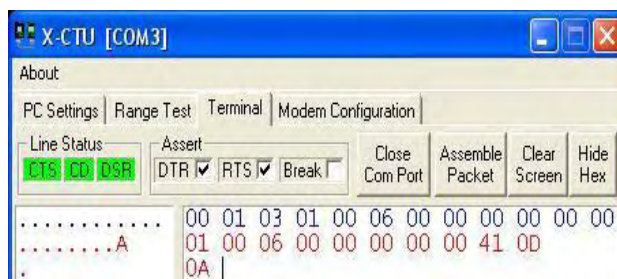


Fonte: Santos Filho (2012)

Como resposta do WTIM_1 à requisição da temperatura, é retornado o valor

apresentado na Figura 44, em que a segunda linha representa o protocolo de resposta e o valor 41_H representa 14 graus Celsius.

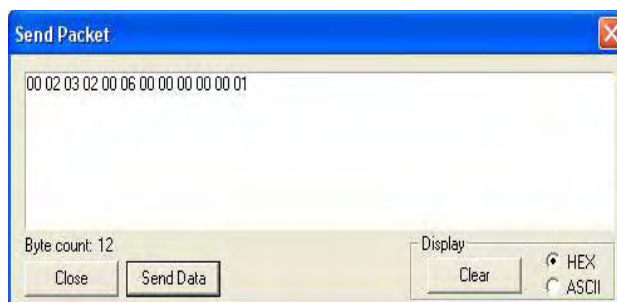
Figura 44 – Protocolo de resposta da temperatura referente ao WTIM_1.



Fonte: Santos Filho (2012)

Para o controle do motor CC presente no WTIM_1, foi utilizado o comando apresentado na Figura 45, em que, no final do comando, o valor 01_H representa ligado.

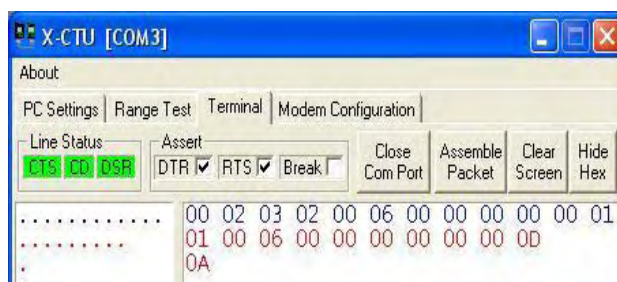
Figura 45 – Protocolo de requisição de controle do motor CC.



Fonte: Santos Filho (2012)

Como resposta do WTIM_1 ao controle do motor CC, é retornado o valor apresentado na Figura 46, em que a segunda linha representa o protocolo de resposta.

Figura 46 – Protocolo de resposta do controle do motor CC referente ao WTIM_1.



Fonte: Santos Filho (2012)

Os protocolos de requisição de temperatura são semelhantes entre os módulos TIMs, realizando apenas a alteração no canal do transdutor. Em relação aos atuadores, os ventiladores e o motor CC trabalham de forma semelhante, em que o último valor do protocolo de requisição 0/1 representa ligado/desligado.

5.6 Considerações Finais Sobre o Capítulo 5

Neste capítulo foi descrito de forma geral, a implementação do TIM e os estados de operação. Além das características dos TIMs, apresentou-se também os módulos desenvolvidos para testes do nó de rede NCAP, sendo, dois módulos utilizando o padrão RS-232 e dois módulos utilizando o padrão ZigBee (IEEE 802.15.4). É importante destacar que os TEDS implementadas nos TIMs encontram-se no Apêndice C e foram descritas baseadas no padrão IEEE 1451.0 - 2007.

Capítulo 6

Desenvolvimento do Nó de Rede IEEE 1451.1

6.1 Introdução

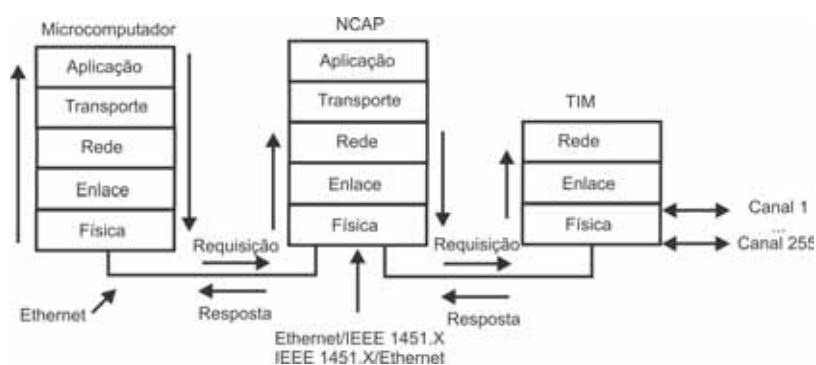
Neste capítulo apresenta-se a implementação de um nó de rede denominado NCAP capaz de receber dados (pacote IP) da rede Ethernet, processar esses dados, enviar para uma de suas interfaces padronizadas e realizar o monitoramento ou o controle dos transdutores inteligentes interligados ao nó. Para o desenvolvimento do nó de rede, foi especificado o *hardware* e o sistema operacional μ Clinux baseado nas características do sistema. Além das características de *hardware* e do sistema operacional, é apresentado o *software* NCAP desenvolvido para realizar o gerenciamento da interface sem fio e com fio baseado no padrão IEEE 1451.

6.2 O Nó IEEE 1451.1 Embarcado

No caso de uma rede de transdutores o nó é um ponto da rede com a capacidade de gerenciar os dados relacionados com uma aplicação envolvendo sensores e atuadores. Neste trabalho, foi desenvolvido um nó de rede utilizando 4 tecnologias de comunicação distintas, SPI, Ethernet, ZigBee e RS-232. A interface SPI realiza a comunicação com a memória *flash* externa (SD card) para o armazenamento dos dados como, página web, logs de acesso aos transdutores e aplicações do NCAP. A comunicação Ethernet é utilizada para realizar o interfaceamento entre o nó de rede e os microcomputadores. A norma IEEE 1451 não define a rede externa, desta maneira, foi utilizada a rede Ethernet. A rede Ethernet neste trabalho tem como função transmitir comandos de controle e monitoramento para o nó de rede NCAP independente do ponto da rede em que se encontra o usuário. Ao receber as informações pela rede Ethernet, o nó de rede processa as informações e converte

para protocolo de rede definido pelo padrão IEEE 1451. O presente trabalho utiliza-se duas interfaces de comunicação, sem fio (ZigBee) e com fio (RS-232), para comunicação com os TIMs definidos de acordo com cada comitê do padrão IEEE 1451. Ao enviar o protocolo de comunicação através de uma das interfaces (com fio ou sem fio) para o TIM, é realizada a leitura ou o controle dos transdutores utilizando um dos canais, que, então, é repassado um protocolo de resposta para o NCAP que converte os dados para padrão Ethernet e retorna para o usuário. Na Figura 47 apresentam-se o esquemático do sistema e a relação com a estrutura de camada do modelo OSI.

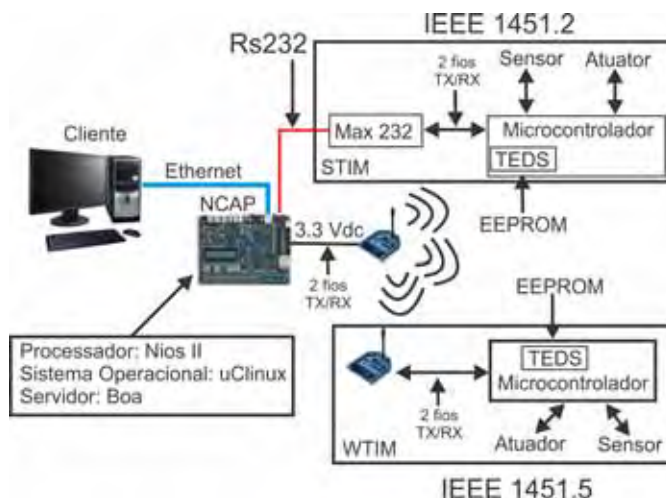
Figura 47 – Sistema baseado na estrutura de camadas do modelo OSI.



Fonte: Santos Filho (2012)

Neste trabalho, o nó de rede padrão IEEE 1451 tem como objetivo obter dados da rede externa Ethernet, processá-los e enviá-los para uma de suas interfaces padronizadas IEEE P1451.2 ou IEEE 1451.5. Na Figura 48 apresenta-se o modelo do sistema desenvolvido com o nó de rede implementado no kit DE 2 da Altera, interface de comunicação externa (Ethernet) e as duas interfaces de comunicação internas (ZigBee e RS-232).

Figura 48 – Esquemático do sistema implementado em laboratório.



Fonte: Santos Filho (2012)

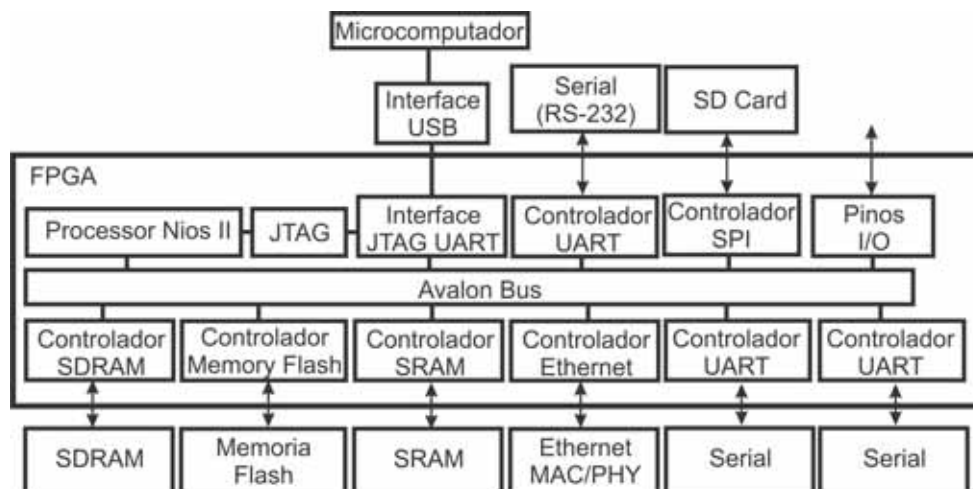
Para o desenvolvimento do nó de rede, o sistema foi dividido em três partes, sendo: *hardware*, sistema operacional embarcado e *software* NCAP. O *hardware* foi definido utilizando o ambiente SoPC Builder de acordo com as características do nó de rede proposto neste trabalho e é descrito na Seção 6.3. O sistema operacional embarcado foi compilado e gerado à imagem baseada nas características de *hardware* e nas aplicações que serão utilizadas para oferecer suporte no desenvolvimento e na configuração do nó de rede. Na Seção 6.4, apresentam-se as configurações definidas para o desenvolvimento do sistema operacional. O *software* NCAP foi desenvolvido em linguagem C, utilizando as mesmas funções de acesso as interfaces de comunicação definidas para microcomputador. A interface com usuário foi desenvolvida em HTML e CGI para que as aplicações feitas em C sejam executadas. É importante ressaltar que o padrão IEEE 1451.1 sugere implementação orientada a objeto, entretanto, não há exigência de que as atuais implementações sejam orientadas a objeto (IEEE, 1999). A descrição das características do nó de rede NCAP implementado neste trabalho é descrita na Seção 6.6.

6.3 Descrição do *Hardware*

O kit DE2 possui diversos componentes permitindo, assim, que projetistas desenvolvam a arquitetura de *hardware* de acordo com as características de seu sistema. Neste trabalho, o *hardware* foi definido segundo as características necessárias para a implementação do nó de rede IEEE 1451.1. Para a arquitetura do nó de rede, foram definidos os seguintes componentes no ambiente SoPC Builder: processador Nios II/f, Avalon-MM *Tristate Bridge*, *Flash Memory Interface*, *SDRAM Controller*, EPCS (*Embedded Programming the Serial Configuration Chip*) *Serial Flash Controller*, JTAG UART, 3 UART (RS-232 Serial Port), *Interval Timer*, *Character LCD*, SEG7_LUT_8, SRAM_16 Bit_512K, SPI (3 Wire Serial), DM9000A (Ethernet) e PIO(Parallel I/O). Na Figura 49 apresenta-se a estrutura do *hardware* definido para o desenvolvimento do nó de rede.

Para implementação do *hardware* apresentada na Figura 49, utilizou-se o ambiente de desenvolvimento Quartus II com a ferramenta SoPC Builder. Na Seção 6.3.1 será apresentada configuração do *hardware* definido no ambiente SoPC Builder e as modificações realizadas no ambiente Quartus II.

Figura 49 – Estrutura para o desenvolvimento do *hardware* do nó de rede IEEE 1451.1.



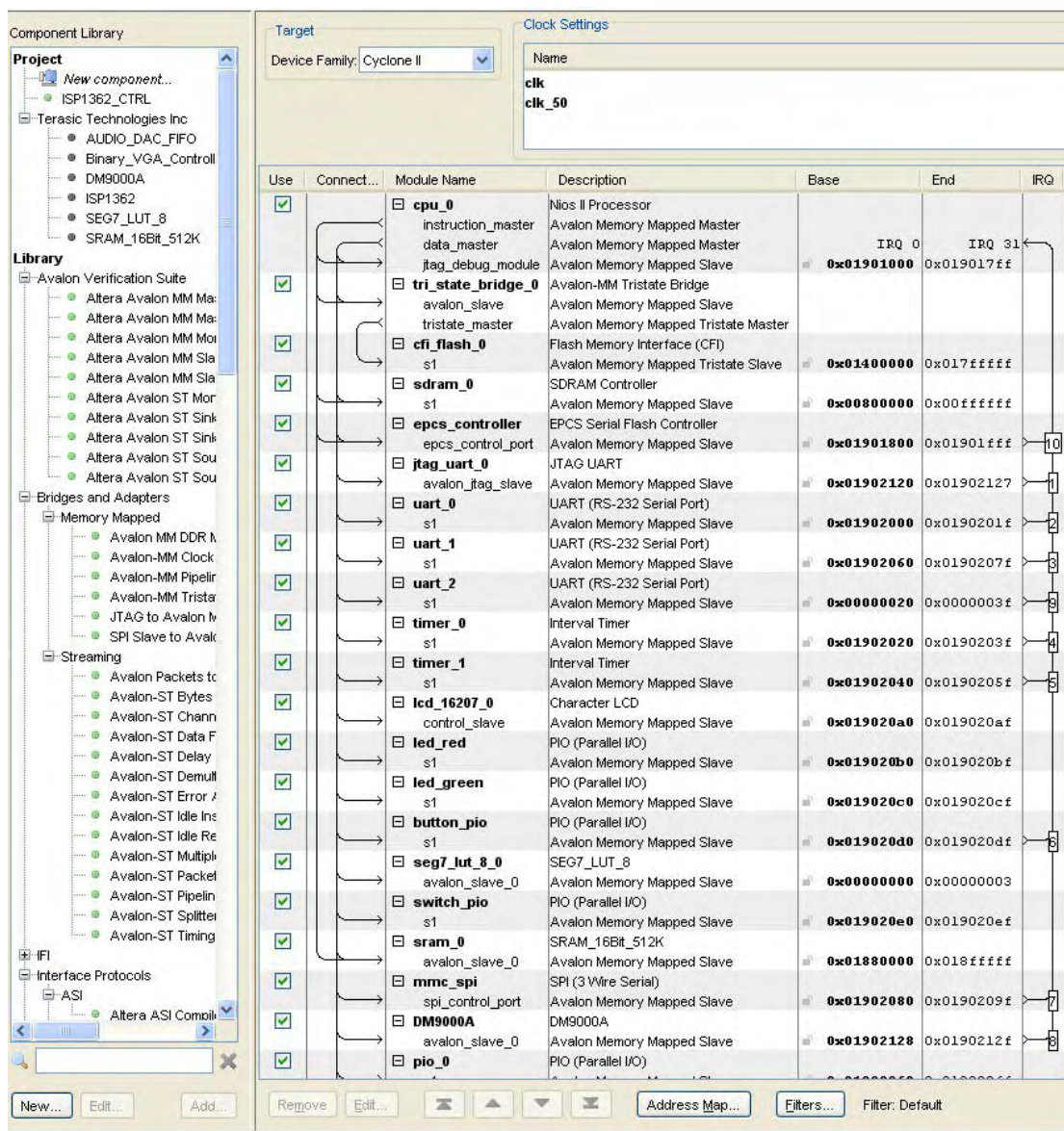
Fonte: Santos Filho (2012)

6.3.1 Definição e Configuração do *Hardware*

O SoPC Builder é uma ferramenta que permite selecionar e configurar cada componente de *hardware* de acordo com a característica do sistema. Na Figura 50, apresentam-se o ambiente SoPC Builder e os blocos definidos para o desenvolvimento do nó de rede embarcado.

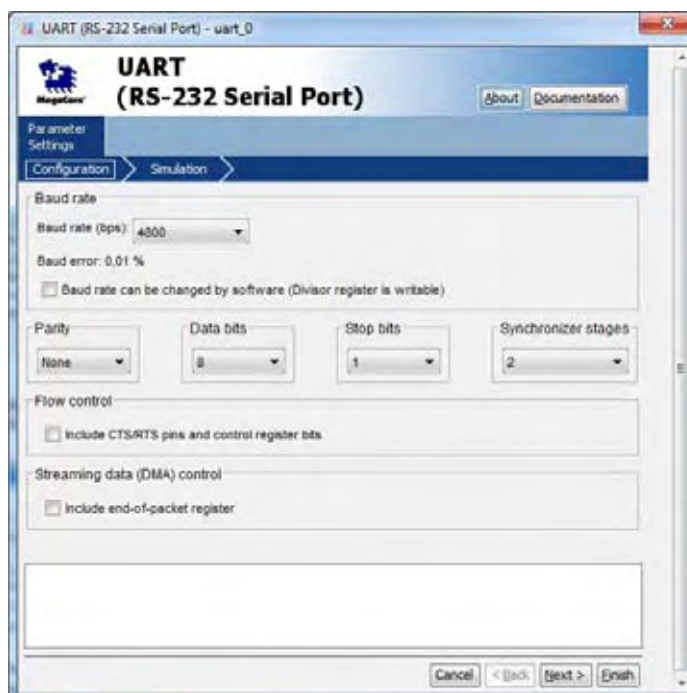
Ao selecionar os blocos, deve-se definir os endereçamentos e as IRQs (*Interrupt Request*), que é definida de forma manual ou automática utilizando a opção “*System*”. Além das configurações das IRQs e os endereçamentos, foram feitas configurações de forma manuais, como por exemplo, o bloco de controle da interface SPI o nome foi alterado de: SPL_0 para mmc_spi. A alteração deve ser realizada para que não ocorra erro na comunicação entre o SD Card com o bloco de controle SPI. Uma outra característica que foi alterada nos blocos de controle foi a velocidade de transmissão das interfaces UART de 9600 bps para 4800 bps. Apesar de ser mais lenta a transmissão em 4800 bps, o sistema não apresentou perda dos dados com a atual configuração definida para o NCAP. Na Figura 51 apresenta-se a interface de configuração da interface UART.

Figura 50 – Ambiente SoPC Builder com os módulos definidos para o desenvolvimento do nó de rede embarcado.



Fonte: Santos Filho (2012)

Figura 51 – Interface de configuração do módulo UART.



Fonte: Santos Filho (2012)

O endereçamento de cada componente é visualizado através da opção “*Address Map*” de acordo com a Figura 52.

Figura 52 – Mapa do endereçamento do projeto.

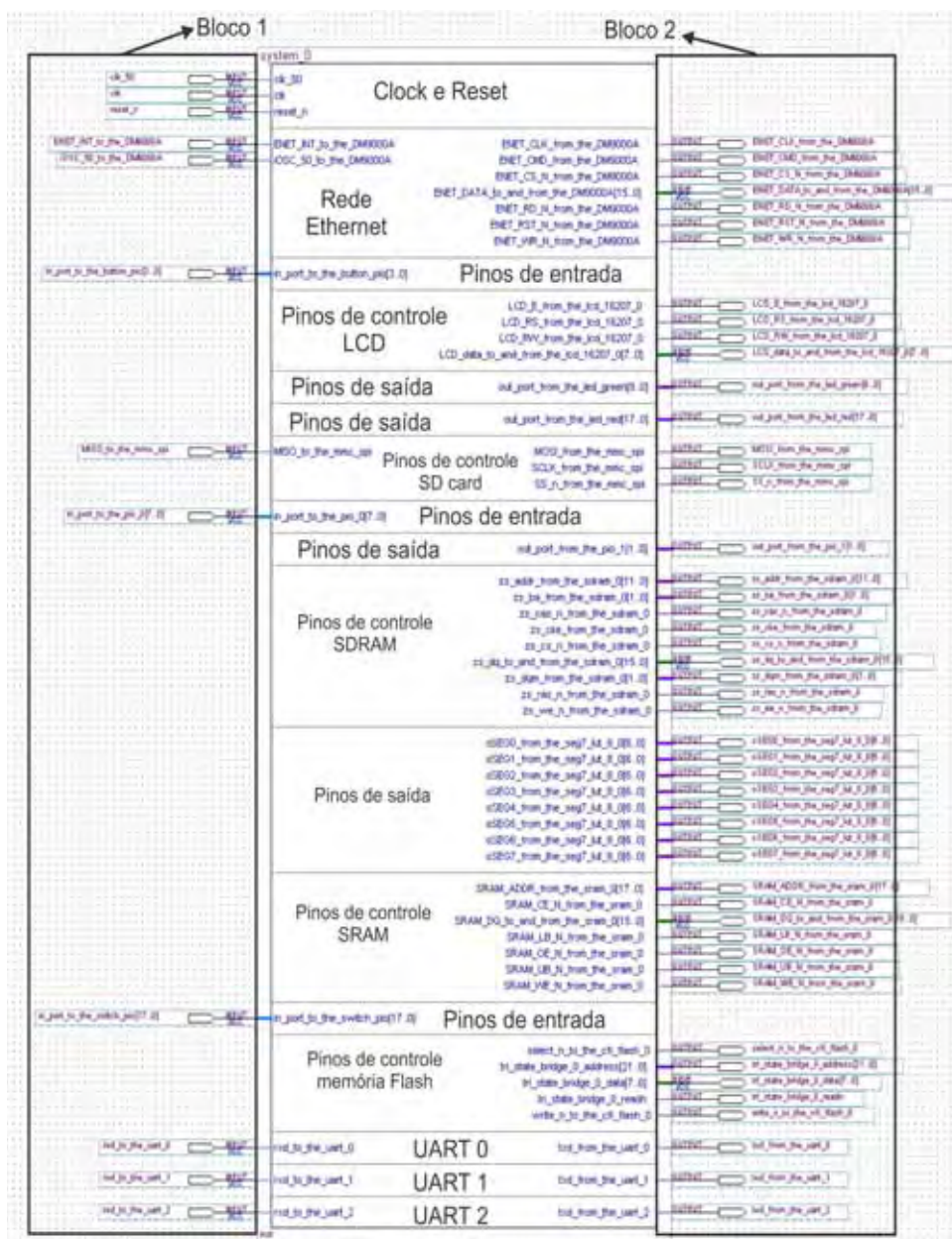
Component	Address Range 1	Address Range 2
cpu_0_instruction_master	0x01901000 - 0x019017ff	0x01901000 - 0x019017ff
cpu_0_data_master		
l1_slave_bridge_0_slave		
cf_flash_0.s1	0x01400000 - 0x017fffff	0x01400000 - 0x017fffff
sdram_0.s1	0x00800000 - 0x00ffffff	0x00800000 - 0x00ffffff
epca_controller_epca_control_port	0x01901800 - 0x01901fff	0x01901800 - 0x01901fff
fpga_uart_0_slave		0x01902120 - 0x01902127
uart_0.s1		0x01902000 - 0x0190201f
timer_0.s1		0x01902020 - 0x0190203f
timer_1.s1		0x01902040 - 0x0190205f
lcd_16207_0_control_slave		0x01902060 - 0x0190206f
led_red.s1		0x01902070 - 0x0190207f
led_green.s1		0x01902080 - 0x0190208f
buttons_p0.s1		0x01902090 - 0x0190209f
switch_p0.s1		0x019020a0 - 0x019020af
sram_0_slave_0	0x01880000 - 0x018fffff	0x01880000 - 0x018fffff
CAM000A_slave_0		0x01902128 - 0x0190212f
p0_0.s1		0x019020fd - 0x019020fe
p0_1.s1		0x019020ff - 0x01902100
uart_1.s1		0x01902040 - 0x0190207f
mmc_ssd_ctrl_control_port		0x01902080 - 0x0190209f
seg7_led_0_slave_0		0x00000000 - 0x00000003
uart_2.s1		0x00000020 - 0x0000003f

Fonte: Santos Filho (2012)

Ao finalizar a estrutura de blocos que será implementada no kit DE2, a opção “*Generate*” gerará toda a estrutura de arquivos e a descrição de *hardware* dos blocos definidos no ambiente SoPC Builder disponibilizando os códigos no ambiente Quartus II. O

hardware gerado pela ferramenta SoPC Builder pode ser visualizado de forma esquemática em um único bloco na Figura 53 ou através da descrição de hardware apresentado na Figura 54. Na Figura 53 no Bloco 1, representa os pinos de entrada de dados no FPGA, na Figura 53 no Bloco 2, os pinos de saída. Os blocos utilizados no projeto foram: *Clock e Reset*, Rede Ethernet, pinos de controle SD card, pinos de controle SDRAM, pinos de controle SRAM, pinos de controle memória Flash e controladores UARTs. Os demais blocos foram inseridos no projeto para futuras implementações.

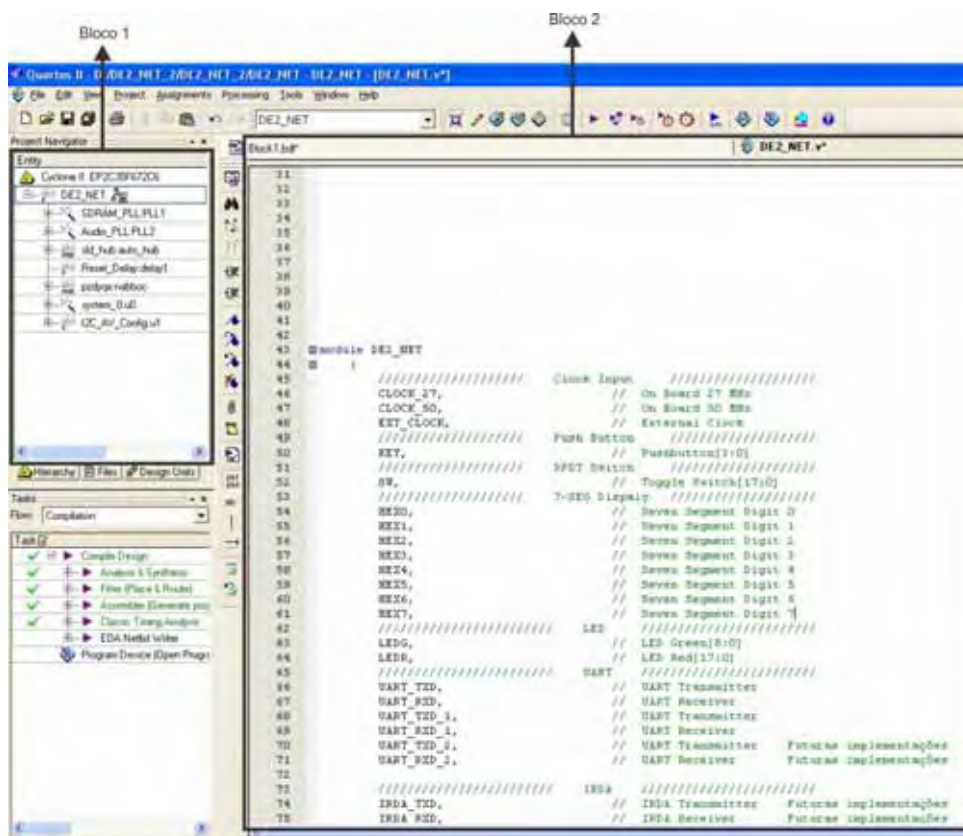
Figura 53 – Bloco dos módulos.



Fonte: Santos Filho (2012)

Na Figura 54, Bloco 1, a estrutura de blocos de acordo com sua hierarquia e, no Bloco 2, o código fonte dos componentes em Verilog.

Figura 54 – Ambiente Quartus II com código verilog.



Fonte: Santos Filho (2012)

O código apresentado na Figura 54 no Bloco 2 representa o arquivo principal do projeto descrito no ambiente SoPC Builder em que são feitas as configurações dos pinos de entrada e saída e as chamadas dos arquivos dos blocos de controle de cada componente. Neste trabalho, foi utilizado o ambiente Quartus II 9.1 web Edition que, ao gerar os códigos dos blocos de controle utilizando a ferramenta SoPC Builder, os pinos de entrada e saída e a chamada das funções dos pinos não são descritos no código do arquivo principal, fazendo com que o projetista tenha que realizar de forma manualmente. Cabe destacar que todos os blocos de controle são criados quando definidos no ambiente SoPC Builder. Na Figura 55 apresenta-se o código que foi adicionado para configurar as variáveis referente aos pinos de entrada e saída da interface UART e, na Figura 56, as funções de entrada e saída.

Para configurar a interface SPI no ambiente Quartus II é necessário adicionar o código:

- .MISO_to_the_mmc_spi(SD_DAT).
- .MOSI_from_the_mmc_spi(SD_CMD).

Figura 55 – Configurações das variáveis dos pinos de entrada e saída da interface UART.

```

65 //////////////////////////////////////////////////// UART //////////////////////////////////////
66 UART_TXD, // UART Transmitter
67 UART_RXD, // UART Receiver
68 UART_TXD_1, // UART Transmitter
69 UART_RXD_1, // UART Receiver
70 UART_TXD_2, // UART Transmitter
71 UART_RXD_2, // UART Receiver
72

```

Configuração dos pinos de entrada e saída para as interfaces UART

Fonte: Santos Filho (2012)

Figura 56 – Funções de entrada e saída da interface UART.

```

489 // the_uart_0
490 .txd_to_the_uart_0(UART_RXD),
491 .txd_from_the_uart_0(UART_TXD),
492
493 //the_uart_1
494 .txd_to_the_uart_1(UART_RXD_1),
495 .txd_from_the_uart_1(UART_TXD_1),
496
497 //the_uart_2
498 .txd_to_the_uart_2(UART_RXD_2),
499 .txd_from_the_uart_2(UART_TXD_2)
500
501 ;

```

Funções das interfaces UART

Fonte: Santos Filho (2012)

- .SCLK_from_the_mmc_spi(SD_CLK).
- .SS_n_from_the_mmc_spi(SD_DAT3).

Na Figura 57 apresenta-se a configuração da interface SPI e Ethernet realizada no ambiente Quartus II.

Figura 57 – Funções das interfaces SPI e Ethernet.

```

388
389 //Configuração da Interface SPI
390 .MISO_to_the_mmc_spi(SD_DAT),
391 .MOSI_from_the_mmc_spi(SD_CMD),
392 .SCLK_from_the_mmc_spi(SD_CLK),
393 .SS_n_from_the_mmc_spi(SD_DAT3),
394
395
396 // Rede Ethernet (DM9000A)
397 .ENET_CLK_from_the_DM9000A(ENET_CLK),
398 .ENET_CMD_from_the_DM9000A(ENET_CMD),
399 .ENET_CS_N_from_the_DM9000A(ENET_CS_N),
400 .ENET_DATA_to_and_from_the_DM9000A(ENET_DATA),
401 .ENET_INT_to_the_DM9000A(ENET_INT),
402
403 .ENET_RD_N_from_the_DM9000A(ENET_RD_N),
404 .ENET_RST_N_from_the_DM9000A(ENET_RST_N),
405 .ENET_WR_N_from_the_DM9000A(ENET_WR_N),
406 .L0C50_to_the_DM9000A(CLOCK_50),
407

```

Funções da Interface SPI

Funções da rede Ethernet

Fonte: Santos Filho (2012)

Para os blocos de entrada e saída (I/O) gerados pela ferramenta SoPC Builder, foi necessário configurar as variáveis e definir a chamada das funções no arquivo principal do projeto no ambiente Quartus II. Na Figura 58 apresenta-se a configuração das variáveis dos pinos de I/O utilizados neste projeto.

Na Figura 59 apresentam-se as funções para acesso aos pinos de entrada e saída.

Ao finalizar o código fonte no ambiente Quartus II, foi necessário configurar os pinos de

Figura 58 – Configurações das variáveis dos pinos de entrada e saída.

```

303 ////////////////////////////////////////////////// GPIO //////////////////////////////////////
304 inout [7:0] GPIO_0; // GPIO Connection 0
305 inout [1:0] GPIO_1; // GPIO Connection 1

```

Fonte: Santos Filho (2012)

Figura 59 – Funções de acesso aos pinos de entrada e saída.

```

441 //pinos de i/o
442 ..in_port_to_the_pio_0(GPIO_0),
443
444 //pio
445 ..out_port_from_the_pio_1(GPIO_1),

```

Fonte: Santos Filho (2012)

entrada e saída de acordo com as características do FPGA. Neste trabalho, foi utilizada o FPGA modelo Cyclone II EP2C35F672C6 e a definição dos pinos para acesso aos periféricos, foram realizados de acordo com o manual, em Terasic (2005). Na Figura 60 apresenta-se a configuração dos pinos do controlador Ethernet, através da opção “*Pin Planner*” do ambiente Quartus II.

Figura 60 – Configuração dos pinos da FPGA referente ao controlador Ethernet.

Node Name	Direction	Location	I/O Bank	WREF Group	I/O Standard	
49	ENET_CS_N	Output	PN_A23	4	04_NO	3.3-V LVTTL (default)
50	ENET_DATA[15]	BiDir	PN_D18	4	04_NO	3.3-V LVTTL (default)
51	ENET_DATA[14]	BiDir	PN_E18	4	04_NO	3.3-V LVTTL (default)
52	ENET_DATA[13]	BiDir	PN_A19	4	04_NO	3.3-V LVTTL (default)
53	ENET_DATA[12]	BiDir	PN_B19	4	04_NO	3.3-V LVTTL (default)
54	ENET_DATA[11]	BiDir	PN_D19	4	04_NO	3.3-V LVTTL (default)
55	ENET_DATA[10]	BiDir	PN_C19	4	04_NO	3.3-V LVTTL (default)
56	ENET_DATA[9]	BiDir	PN_A20	4	04_NO	3.3-V LVTTL (default)
57	ENET_DATA[8]	BiDir	PN_B20	4	04_NO	3.3-V LVTTL (default)
58	ENET_DATA[7]	BiDir	PN_B15	4	04_NO	3.3-V LVTTL (default)
59	ENET_DATA[6]	BiDir	PN_B16	4	04_NO	3.3-V LVTTL (default)
60	ENET_DATA[5]	BiDir	PN_A17	4	04_NO	3.3-V LVTTL (default)
61	ENET_DATA[4]	BiDir	PN_B17	4	04_NO	3.3-V LVTTL (default)
62	ENET_DATA[3]	BiDir	PN_A18	4	04_NO	3.3-V LVTTL (default)
63	ENET_DATA[2]	BiDir	PN_B18	4	04_NO	3.3-V LVTTL (default)
64	ENET_DATA[1]	BiDir	PN_C17	4	04_NO	3.3-V LVTTL (default)
65	ENET_DATA[0]	BiDir	PN_D17	4	04_NO	3.3-V LVTTL (default)
66	ENET_INT	Input	PN_B21	4	04_NO	3.3-V LVTTL (default)
67	ENET_RD_N	Output	PN_A22	4	04_NO	3.3-V LVTTL (default)
68	ENET_RST_N	Output	PN_B23	4	04_NO	3.3-V LVTTL (default)
69	ENET_WR_N	Output	PN_B22	4	04_NO	3.3-V LVTTL (default)
70	ENET_CLOCK	Input	PN_P26	6	06_NO	3.3-V LVTTL (default)

Fonte: Santos Filho (2012)

Para realizar a comunicação com os TIMs utilizando interface com fio, foi definida a entrada e a saída do controlador denominado UART_0. A saída do bloco UART_0 foi conectado na entrada do circuito integrado MAX232 e a entrada na saída do MAX232 do próprio kit DE2. Para a interface sem fio, foi utilizado o controlador UART_1 definindo-se a entrada e a saída no barramento de pinos (IDE_0) do kit DE2, denominado JP0. A entrada e a saída da UART_1 são conectadas ao módulo XBee-Pro, que realiza o empacotamento dos dados e envia, através da interfaces sem fio, utilizando o padrão IEEE 802.15.4. O controlador UART_2 refere-se a futuras implementações que podem ser realizadas no nó de rede IEEE 1451.1, utilizando interface UART. Na Figura 61,

apresentam-se as configurações dos pinos de entrada e saída.

Figura 61 – Configuração dos pinos dos controladores UART_0, UART_1 e UART_2.

327	BP	UART_RXD	Input	PIN_C25	5	B5_N0	3.3-V LVTTL (default)
328	BP	UART_RXD_1	Input	PIN_K19	5	B5_N1	3.3-V LVTTL (default)
329	BP	UART_RXD_2	Input				3.3-V LVTTL (default)
330	BP	UART_TXD	Output	PIN_B25	5	B5_N0	3.3-V LVTTL (default)
331	BP	UART_TXD_1	Output	PIN_K21	5	B5_N1	3.3-V LVTTL (default)
332	BP	UART_TXD_2	Output				3.3-V LVTTL (default)

Fonte: Santos Filho (2012)

As interfaces UART apresentadas na Figura 61 são representadas como:

- **UART** - UART_RXD (entrada), UART_TXD (saída).
- **UART_1** - UART_RXD_1 (entrada), UART_TXD_1 (saída).
- **UART_2** - UART_RXD_2 (entrada), UART_TXD_2 (saída).

A comunicação com o SD Card foi realizada através da interface SPI e os pinos de entrada e saída do bloco de controle SPI foram definidos de acordo com a Figura 62

Figura 62 – Configuração dos pinos do controlador SPI.

251	BP	SD_CLK	Output	PIN_AD23	6	B6_N1	3.3-V LVTTL (default)
252	BP	SD_CMD	Input	PIN_Y21	6	B6_N1	3.3-V LVTTL (default)
253	BP	SD_DAT0	Input	PIN_AD24	6	B6_N1	3.3-V LVTTL (default)
254	BP	SD_DAT3	Input	PIN_AC23	6	B6_N1	3.3-V LVTTL

Fonte: Santos Filho (2012)

Ao finalizar as configurações dos pinos de entrada e saída dos blocos controladores, foi realizada a compilação do projeto. Ao término da compilação do projeto, foram apresentados ao projetista os recursos que foram utilizados de acordo com a Figura 63.

Figura 63 – Análise dos componentes utilizados no projeto.

```

Flow Status                Successful - Wed Jul 27 15:22:25 2011
Quartus II Version         9.1 Build 222 10/21/2009 SJ Full Version
Revision Name              DE2_NET
Top-level Entity Name      DE2_NET
Family                     Cyclone II
Device                     EP2C35F672C6
Timing Models              Final
Met timing requirements     Yes
Total logic elements        5,769 / 33,216 ( 17 % )
  Total combinational functions  4,903 / 33,216 ( 15 % )
  Dedicated logic registers    3,479 / 33,216 ( 10 % )
Total registers            3596
Total pins                  367 / 475 ( 77 % )
Total virtual pins         0
Total memory bits          75,904 / 483,840 ( 16 % )
Embedded Multiplier 9-bit elements  4 / 70 ( 6 % )
Total PLLs                  1 / 4 ( 25 % )

```

Fonte: Santos Filho (2012)

uma lista de produtos podendo ser selecionada de acordo com a característica do seu *hardware*. Na Figura 65 apresentam-se as opções para inserção do fabricante e, na Figura 66, apresentam-se alguns exemplos dos fabricantes no qual o μ Clinux oferece suporte.

Figura 65 – Interface de definição do *hardware*.

```

|-----|
x  --- Select the Vendor you wish to target                               x
x  Vendor (Altera) --->                                                 x
x  --- Select the Product you wish to target                             x
x  Altera Products (nios2) --->                                         x
x  |-----|

```

Fonte: Santos Filho (2012)

Figura 66 – Exemplo de suporte oferecido pelo μ Clinux aos diversos fabricantes.

```

|-----|
x  ( ) Actiontec                                                         x
x  ( ) ADI                                                               x
x  ( ) Akizuki                                                           x
x  (X) Altera                                                            x
x  (●) Apple                                                             x
x  ( ) Arcturus                                                         x
|-----|

```

Fonte: Santos Filho (2012)

Na Figura 67, apresenta-se a interface de configuração das bibliotecas do sistema operacional μ Clinux, a qual possui quatro opções:

- **Default all settings** - bibliotecas pré-definidas pelo compilador;
- **Customize Kernel Settings** - configurações do Kernel definida pelo usuário;
- **Customize Application/Library Settings** - configurações dos aplicativos e bibliotecas pelo usuário;
- **Update Default Vendor Settings** - atualização padrão das configurações do vendedor.

Figura 67 – Opções de configuração do μ Clinux.

```

|-----|
x  --- Kernel is linux-2.6.x                                             x
x  Libc Version (None) --->                                             x
x  [ ] Default all settings (lose changes)                               x
x  [ ] Customize Kernel Settings                                         x
x  [ ] Customize Application/Library Settings                             x
x  [ ] Update Default Vendor Settings                                     x
x  |-----|
x  |-----|
x  |-----|
x  |-----|
|-----|

```

Fonte: Santos Filho (2012)

Tabela 11 – Tabela de comandos e aplicativos definidos no projeto.

Bloco de Ferramentas	Sub-bloco	Componente
BusyBox	Coreutils	cat, chown, cp, date, df, intall, mkdir, pwd, rmdir, who, chmod, du, cut, dd, echo, ls, mv, ram, tty, whoime
	Consoles Utilities	clear, reset, chvt
	Debian Utilities	Pipe progress
	Editors	ed, vi
	Find Utilities	find, grep
	Init Utilities	init, power, off, halt, reboot
	Linux Module Utilities	insmode, rmmmod, lsmod, modprobe
	Linux System Utilities	dmesg, fdisk, freeramdisk, mount, umount
	Network Utilities	arp, ftpget, ftpput, httpd, ifconfig, inetd, IP, netstat, ping, route, wget
	Process Utilities	free, kill, ps, top, uptime
	Shell	msh

Fonte: Santos Filho (2012)

Ao finalizar a definição dos componentes, processador e memória, é executado o comando `#make` para compilar e gerar a imagem do sistema operacional embarcado. Após finalizar a compilação, será criado um arquivo imagem dentro do diretório `# ../uClinux-dist/images/` denominado de `zImage`. O arquivo `zImage` é um pacote de dados que contém todas as ferramentas selecionadas pelo projetista no ambiente `uClinux-dist` e que, ao ser transferida para o kit DE2, é descompactada e feitas as configurações necessárias do sistema operacional embarcado μ Clinux.

Na Seção 6.4.2, será apresentada a transferência do *hardware* e do sistema operacional μ Clinux para o kit DE2 e a inicialização do sistema.

6.4.2 Transferência do *Hardware* e da Imagem para Kit de Desenvolvimento DE 2

A transferência do *hardware* e da imagem do sistema operacional μ Clinux é realizada através do ambiente Quartus II e em modo de comando disponibilizado junto à EDA (*Environmental Development Association*) de desenvolvimento Nios II. O ambiente em modo de comando assemelha-se com o modo Shell do Linux e ao modo de comando do Windows possuindo os comandos básicos de ambos os sistemas. Neste trabalho, utilizou-se

Figura 72 – Opção de seleção do processador e memória de armazenamento.

```

--- Please select a device to upload the kernel to:
(1) cfi_flash_0
    Class: altera_avalon_cfi_flash
    Size: 1048576 bytes

Selection: 1

--- Please select a device to execute kernel from:
(1) sram_0
    Class: user_logic_SRAM_16Bits_512K
    Size: 524288 bytes

(2) sdram_0
    Class: altera_avalon_new_sdram_controller
    Size: 8388608 bytes

(3) epcs_controller
    Class: altera_avalon_epcs_flash_controller
    Size: 2048 bytes

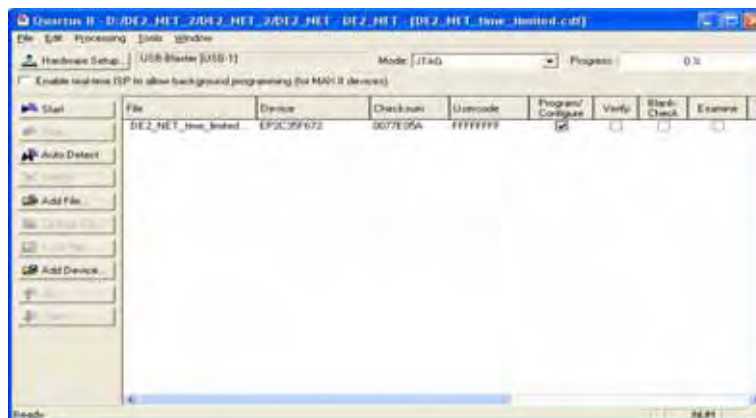
Selection: 2

```

Fonte: Santos Filho (2012)

o ambiente Quartus II para programar o *hardware* e o modo de comando para transferência da imagem do sistema operacional. A transferência da parte do *hardware* para o FPGA é realizada através da opção “*Programmer*” no ambiente Quartus II que disponibilizará uma interface usuário de acordo com a Figura 73.

Figura 73 – Interface usuário para programação do projeto na FPGA.

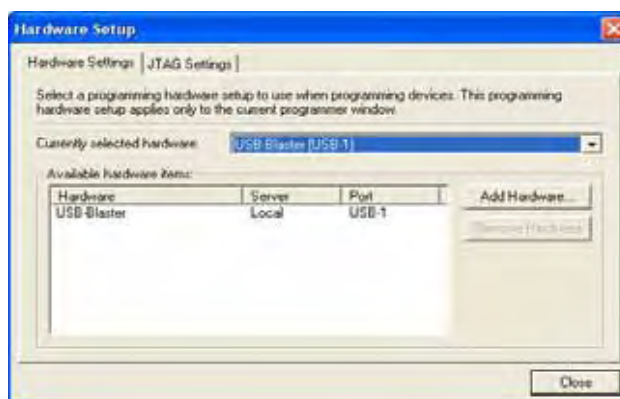


Fonte: Santos Filho (2012)

Para definir a interface de comunicação entre o microcomputador e o kit DE2, é utilizada a opção “*Hardware Setup*” disponibilizado na Figura 73. Ao selecionar a opção “*Hardware Setup*”, é disponibilizada uma interface usuário em que permite selecionar a interface de comunicação com o kit DE2 de acordo com a Figura 74. Neste trabalho utilizou-se a interface USB, no qual o *driver* do dispositivo pode ser localizado dentro do diretório #C:/altera/80/quartus/drivers, definido como padrão no sistema operacional Windows. Ao finalizar a configuração da interface de comunicação com o kit DE2, é apresentada novamente a interface usuário de programação apresentada na Figura 73. O

arquivo de transferência “.sof” é reconhecido de forma automática pelo ambiente Quartus II e referenciado para o projetista. Ao clicar na opção “Start”, inicializa-se o processo de programação da FPGA.

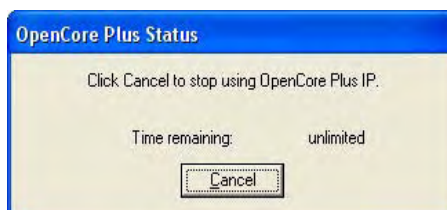
Figura 74 – Definição da interface de comunicação entre o microcomputador e o kit DE2.



Fonte: Santos Filho (2012)

As ferramentas e as configurações do *hardware* para o desenvolvimento do nó de rede foram desenvolvidas utilizando Quartus II web Edition, versão do Quartus II que, ao carregar o projeto na FPGA, mantém a comunicação com o microcomputador, através da interface JTAG UART utilizando o cabo USB. Ao retirar o cabo USB, o processador Nios II para a execução das instruções e o sistema operacional embarcado para a execução dos processos, devido à versão estudante do ambiente Quartus II utilizado neste trabalho. Na Figura 75, apresenta-se a interface usuário disponibilizada pelo ambiente Quartus II, após a transferência do projeto para a FPGA e, na Figura 76, a mensagem referente à licença.

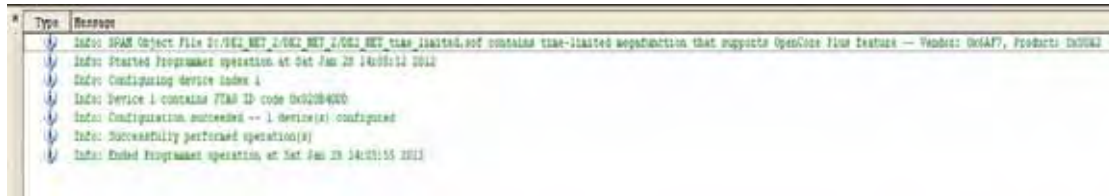
Figura 75 – Interface usuário da licença do ambiente Quartus II.



Fonte: Santos Filho (2012)

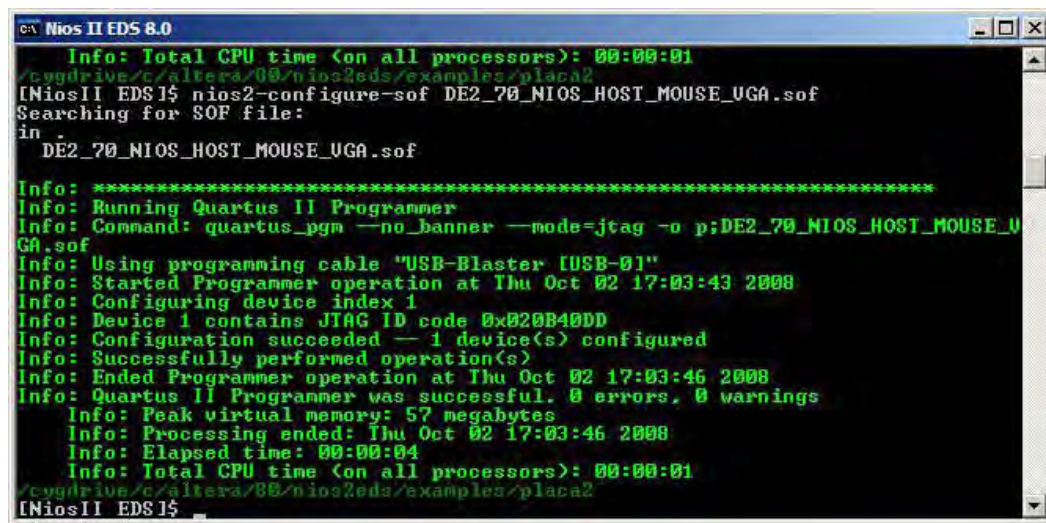
Em modo de comando, para realizar a transferência do bloco de *hardware* gerado pelo ambiente SoPC Builder, apresentado na Figura 53, deve-se realizar o seguinte comando: `#nios2-configure-sof <nome_do_arquivo.sof>`. Como confirmação da transferência do arquivo, são apresentadas informações de erro, tempo de transferência dos dados e a confirmação do processo realizado com sucesso. Na Figura 77 apresenta-se a transferência do *hardware*, utilizando o modo de comando Nios II.

Figura 76 – Mensagem apresentada pelo Quartus II referente à licença.



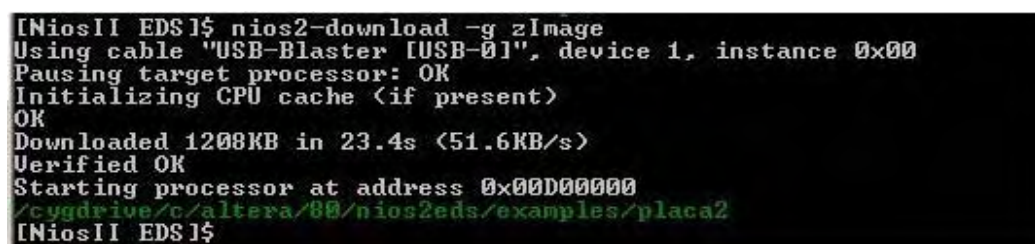
Fonte: Santos Filho (2012)

Figura 77 – Exemplo de transferência do arquivo sof.



Fonte: Santos Filho (2012)

Para transferir a imagem do sistema operacional μ Clinux para o kit DE2, deve ser executado o comando `#nios2-download -g <diretório><nome_da_imagem>`. O parâmetro “-g” indica o ponto de execução do processador, após ser carregado na memória. Na Figura 78, apresenta-se a transferência do arquivo zImage referente à imagem do kernel gerada pelo compilador *software* uClinux-dist. Após o envio da imagem do kernel, é apresentado ao usuário uma mensagem de confirmação indicando que o processo foi realizado com sucesso.

Figura 78 – Transferência da imagem do μ Clinux para o kit Nios II.

Fonte: Santos Filho (2012)

Para inicializar o sistema operacional μ Clinux é realizado o comando `#nios2-terminal`.

O comando iniciará o sistema operacional embarcado, apresentando para o usuário suas configurações de *hardware* de acordo com Figura 79.

Figura 79 – Sistema operacional μ Clinux sendo inicializado.

```

c:\ Nios II EDS 8.0
Verified OK
Starting processor at address 0x00D00000
/cygdrive/c/altera/00/nios2eds/examples/placa2
NiosII EDS15 nios2-terminal
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-01]", device 1, instance 0
nios2-terminal: <Use the IDE stop button or Ctrl-C to terminate>

Uncompressing Linux... Ok, booting the kernel.
Linux version 2.6.19-uc1-g7e2efdd1-dirty (thomas@darkstar.wytron.com.tw) (gcc ve
rsion 3.4.6) #5 PREEMPT Wed Mar 12 10:38:03 CST 2008

uClinux/Nios II
Altera Nios II support (C) 2004 Microtronix Datacom Ltd.

setup_arch: No persistant network settings signature at 003F0000
Built 1 zonelists. Total pages: 2032
Kernel command line:
PID hash table entries: 32 (order: 5, 128 bytes)
Dentry cache hash table entries: 1024 (order: 0, 4096 bytes)
Inode-cache hash table entries: 1024 (order: 0, 4096 bytes)
Memory available: 5744k/8192k RAM, 0k/0k ROM (1473k kernel code, 694k data)
Mount-cache hash table entries: 512

```

Fonte: Santos Filho (2012)

Na Seção 6.5, serão apresentadas as características do ambiente μ Clinux e as ferramentas utilizadas para o desenvolvimento do nó de rede IEEE 1451.1 e o suporte ao administrador da rede.

6.5 Ambiente μ Clinux

O μ Clinux implementado neste trabalho assemelha-se aos Linux convencionais utilizados em microcomputadores em modo de comando, possuindo os comandos básicos de manipulação de arquivos, configuração de interfaces de rede, hierarquia de diretórios, arquivos de configuração do sistema e acesso aos periféricos utilizando programação de alto nível. Nesta seção, serão apresentadas as principais características do sistema operacional μ Clinux e onde se encaixam dentro do contexto do trabalho.

Ao inicializar o sistema operacional μ Clinux utilizando o modo de comando, são realizadas as configurações preliminares do sistema, como ativar a interface SPI, interfaces UART, memórias, rede Ethernet, diretórios e arquivos. Ao carregar o sistema, é apresentada ao usuário a interface em modo de comando de acordo com a Figura 80.

As informações sobre *hardware* e do sistema operacional μ Clinux podem ser obtidas através de comandos no terminal. Para visualizar as características de *hardware* referente à CPU é executado o comando: `#cat /proc/cpuinfo`. Na Figura 81 apresentam-se as características referentes à CPU.

Figura 80 – Sistema operacional μ Clinux em modo de comando.

```

c:\ Nios II EDS 8.0
Command: mkdir /var/tmp
Command: mkdir /var/log
Command: mkdir /var/run
Command: mkdir /var/lock
Command: mkdir /var/empty
Command: ifconfig lo 127.0.0.1
Command: route add -net 127.0.0.0 netmask 255.0.0.0 lo
Command: cat /etc/motd
Welcome to

      _ _ _ _ _
     / / / / /
    / / / / /
   / / / / /
  / / / / /
 / / / / /
/ / / / /

For further information check:
http://www.uclinux.org/

Execution Finished, Exiting

Sash command shell <version 1.1.1>
/>

```

Fonte: Santos Filho (2012)

Figura 81 – Informações da CPU visualizadas através do ambiente μ Clinux.

```

/> cat /proc/cpuinfo
CPU:          NIOS2
MMU:          none
FPU:          none
Clocking:     100.0MHz
BogoMips:     44.74
Calibration:  22374400 loops
/>

```

Fonte: Santos Filho (2012)

Para visualizar as interrupções, é realizado o comando: `#cat /proc/interrupts`. Na Figura 82 apresentam-se as interrupções referentes às interfaces de comunicação.

Figura 82 – Interrupções visualizadas através do ambiente μ Clinux.

```

/> cat /proc/interrupts
CPU0
1: 131 NIOS2-INTC JTAGUART
2: 0 NIOS2-INTC UART
3: 0 NIOS2-INTC UART
4: 113099 NIOS2-INTC timer
7: 22074 NIOS2-INTC altspi
8: 47 NIOS2-INTC eth0
9: 0 NIOS2-INTC UART
10: 0 NIOS2-INTC altspi
/>

```

Fonte: Santos Filho (2012)

A versão do sistema operacional μ Clinux é obtida através do comando: `#cat /proc/version`. Na Figura 83 apresenta-se a versão do sistema operacional.

A lista dos dispositivos é visualizada através do comando: `#cat /proc/devices`. Na Figura 84 apresentam-se os dispositivos do nó de rede NCAP.

Figura 83 – Versão do sistema operacional μ Clinux.

```

/> cat /proc/version
uClinux version 2.6.30 (root@stim) (gcc version 3.4.6) #272 PREEMPT Fri Jun 3 17
:08:12 BRT 2011
/>

```

Fonte: Santos Filho (2012)

Figura 84 – Dispositivos do nó de rede.

```

/> cat /proc/devices
Character devices:
1 mem
2 pty
3 tty
4 ttyS
5 /dev/tty
5 /dev/console
5 /dev/ptmx
10 misc
128 ptm
136 pts
232 ttyJ

Block devices:
259 blkext
179 mmc
/>

```

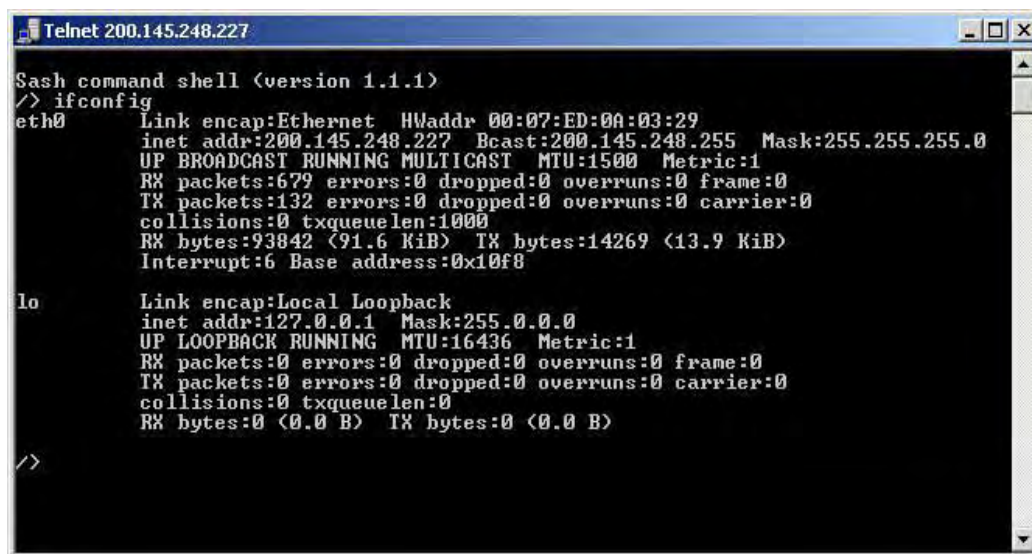
Fonte: Santos Filho (2012)

Para realizar a configuração da rede Ethernet, deve-se primeiramente definir o endereço MAC através do comando `#ifconfig eth0 hw ether <endereço MAC com 48 bits>`. O endereço MAC é o endereço físico da estação e possui 48 bits representados em hexadecimal. Este endereço faz parte da camada de Enlace e é responsável pelo controle de acesso de cada estação à rede Ethernet. Para atribuir endereçamento IP é realizado o seguinte comando `#ifconfig eth0 <número_do_IP> netmask <número_da_máscara_de_rede>` em que o parâmetro “eth0” representa a identificação da interface de rede a que será atribuído o endereço IP. Após realizar o comando para a atribuição do endereço IP, pode-se verificar a configuração através do comando `#ifconfig` de acordo com a Figura 85.

Com a rede Ethernet configurada, o μ Clinux disponibiliza para o administrador serviços de FTP, Telnet e servidor web. Os serviços podem ser inicializados através da inicialização do sistema operacional ou utilizando o terminal do μ Clinux. No Apêndice A.2.3 apresentam-se as configurações do servidor web Boa, servidor FTP e Telnet utilizados no nó de rede IEEE 1451.1. Na Figura 86 apresentam-se os comandos do servidor FTP implementado no kit DE2.

Além do modo terminal, a transferência dos arquivos pode ser realizada através do navegador web utilizando o comando `#ftp://<endereço IP>` no campo de endereçamento. Ao finalizar o comando, seleciona-se o arquivo e arrasta para o navegador que realizará a transferência para o kit DE2. Na Figura 87 apresenta-se o navegador web acessando o servidor FTP no kit DE2.

Figura 85 – Configuração da rede Ethernet.



```

Telnet 200.145.248.227
Sash command shell (version 1.1.1)
/> ifconfig
eth0      Link encap:Ethernet  HWaddr 00:07:ED:0A:03:29
          inet addr:200.145.248.227  Bcast:200.145.248.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:679  errors:0  dropped:0  overruns:0  frame:0
          TX packets:132  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0  txqueuelen:1000
          RX bytes:93842 (91.6 KiB)  TX bytes:14269 (13.9 KiB)
          Interrupt:6  Base address:0x10f8

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0  errors:0  dropped:0  overruns:0  frame:0
          TX packets:0  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0  txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

/>

```

Fonte: Santos Filho (2012)

Figura 86 – Comandos para manipulação do servidor FTP.



```

Nios II EDS 8.0
ascii      form      mode      quote     system
bell       get       modtime   recu      sunique
binary     glob     mput     reget     tenex
bye        hash     newer    rstatus   tick
case       help     nmap     rhelp     trace
cd         idle     nlist    rename    type
cdup       image    ntrans   reset     user
chmod      lcd      open     restart  unmask
close      ls       prompt   rmdir    verbose
cr         macdef   passive  runique  ?
delete     mdelete proxy     send

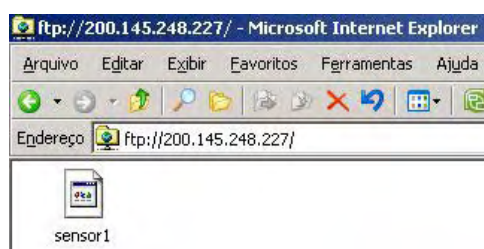
ftp> help
Commands may be abbreviated.  Commands are:

!          debug     mdir     sendport  site
$          dir       mget     put        size
account   disconnect mls      pwd        status
append    exit      mkdir    quit       struct
ascii     form      mode     quote     system
bell      get       modtime  recu      sunique
binary    glob     mput     reget     tenex
bye       hash     newer    rstatus   tick
case      help     nmap     rhelp     trace
cd        idle     nlist    rename    type
cdup      image    ntrans   reset     user
chmod     lcd      open     restart  unmask
close     ls       prompt   rmdir    verbose
delete    mdelete proxy     runique  ?
ftp>

```

Fonte: Santos Filho (2012)


Figura 87 – Transferência de arquivo para o servidor FTP.



Fonte: Santos Filho (2012)

Para acessar o kit DE2 e realizar as configurações necessárias do sistema, é disponibilizado para o usuário serviço de acesso remoto por Telnet. Através do acesso remoto por Telnet, o administrador possui todo o controle e acesso aos arquivos de configuração, independente de sua localização através da Internet, bastando apenas ter acesso à Internet. Na Figura 88, apresenta-se o acesso ao kit DE2, através do serviço de acesso remoto Telnet.

Figura 88 – Arquivos de configuração do ambiente μ Clinux.



```

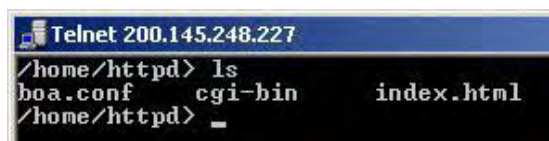
Telnet 200.145.248.227
/> cd /etc/
/etc> ls
TZ          default      ftpwelcome  hosts       mime.types  rc
boa.conf   dhcp        group       inetd.conf  motd        services
config     ftpusers    host.conf   inittab     passwd      version
/etc>

```

Fonte: Santos Filho (2012)

Para armazenamento das páginas web, foi utilizado o servidor Boa. A configuração do servidor Boa é descrita no Apêndice A.2.3. Na Figura 89, apresenta-se o terminal do μ Clinux e o diretório de armazenamento das páginas web.

Figura 89 – Diretório de armazenamento das páginas web no ambiente μ Clinux.



```

Telnet 200.145.248.227
/home/httpd> ls
boa.conf  cgi-bin    index.html
/home/httpd> _

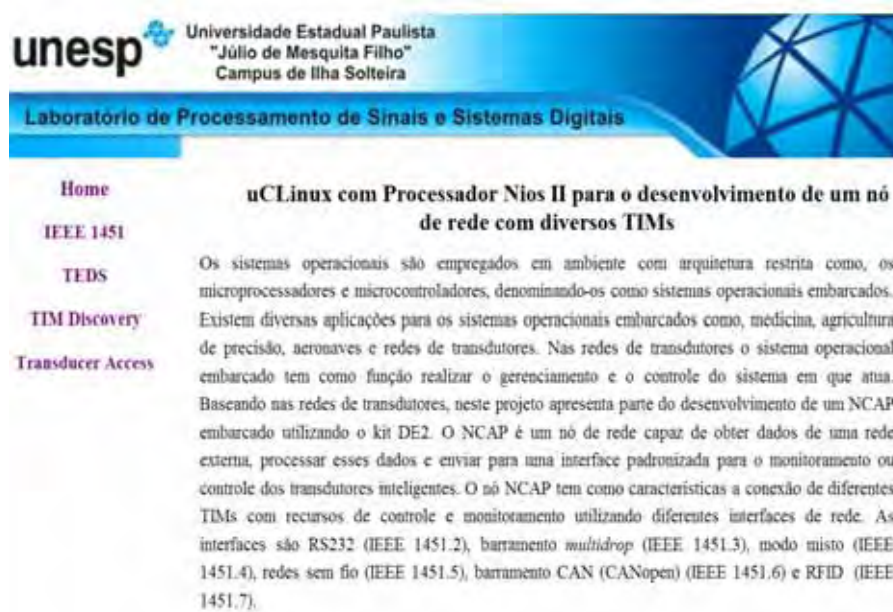
```

Fonte: Santos Filho (2012)

Na Figura 90 apresenta-se o navegador acessando a página web principal disponibilizada pelo servidor Boa.

6.6 *Software* NCAP

O *software* NCAP foi desenvolvido em linguagem C e HTML utilizando o protocolo de comunicação CGI. O padrão IEEE 1451.1 apresenta exemplo de implementação do NCAP utilizando orientação a objeto, porém, não demonstra para aplicações de programação estruturada. Neste contexto, para o desenvolvimento do trabalho foi elaborada uma nova estrutura de representação do NCAP em que cada classe sugerida pelo padrão foi representada por um arquivo e cada arquivo recebeu um nome baseado na norma do padrão IEEE 1451.1, como por exemplo: IEEE1451 _ <nome do arquivo>. Todos os arquivos HTML do NCAP foram armazenados no diretório raiz “/mnt/htdocs/” do servidor web Boa e os arquivos executáveis gerados pela linguagem C foram armazenados

Figura 90 – Página web armazenada no μ Clinux.

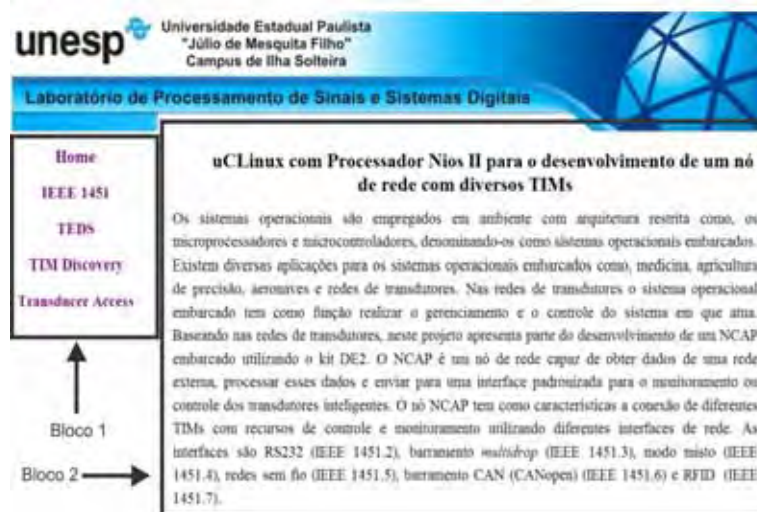
Fonte: Santos Filho (2012)

no diretório “/mnt/htdocs/cgi-bin”. É importante destacar que os códigos descritos em linguagem C devem ser compilados, utilizando bibliotecas do μ Clinux-dist. O comando de compilação é descrito no Apêndice A.2.4.

O NCAP descrito neste trabalho utiliza 2 interfaces, com fio e sem fio. Para acesso às interfaces, o NCAP possui arquivos comuns que realizam as mesmas funcionalidades, sendo: IEEE1451_Block, IEEE1451_FunctionBlock, IEEE1451_Parameter, IEEE1451_TransducerBlock. Os arquivos para acesso às interfaces foram definidos como: IEEE1451_InterfaceConfigZigBee e IEEE1451_InterfaceConfigRS232, pois cada interface possui características próprias para realizar a comunicação.

As páginas web armazenadas no nó de rede NCAP são acessadas através do navegador web inserindo o endereçamento IP do nó na barra de endereçamento. O endereço IP definido para o nó de rede desenvolvido neste trabalho, foi “192.168.1.10” podendo ser alterado de acordo com a configuração da rede em que se encontra. Ao confirmar o endereço IP, o microcomputador realiza a requisição da página web, armazenada no servidor Boa e retorna ao cliente com o conteúdo da página apresentando no navegador a interface principal (Home) do NCAP de acordo com a Figura 91.

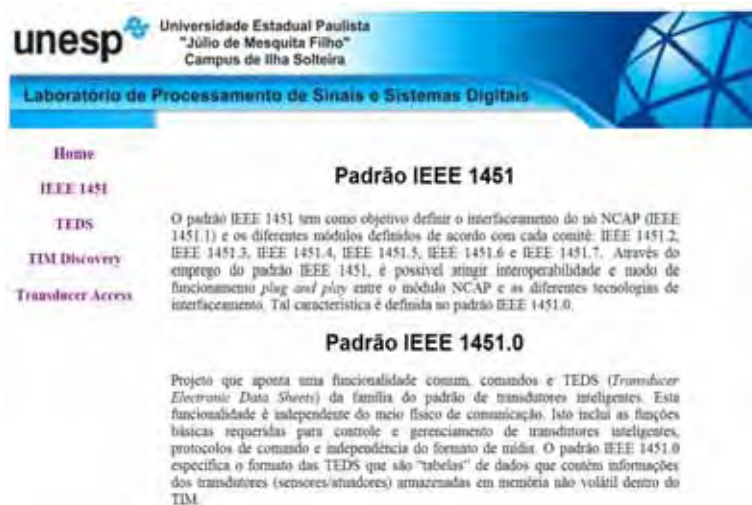
As opções de descrição dos padrões, descrição dos TEDS e acesso aos transdutores são apresentadas na Figura 91 no Bloco 1 e são acessadas através dos links: Home, IEEE 1451, TEDS e Transducer access. O conteúdo de cada opção é apresentado na Figura

Figura 91 – Interface principal do *software* NCAP.

Fonte: Santos Filho (2012)

91 no Bloco 2. Na Figura 92, apresenta-se o conteúdo da opção “IEEE 1451”, em que disponibiliza de forma resumida cada comitê e suas especificações.

Figura 92 – Interface de descrição dos comitês do padrão IEEE 1451.

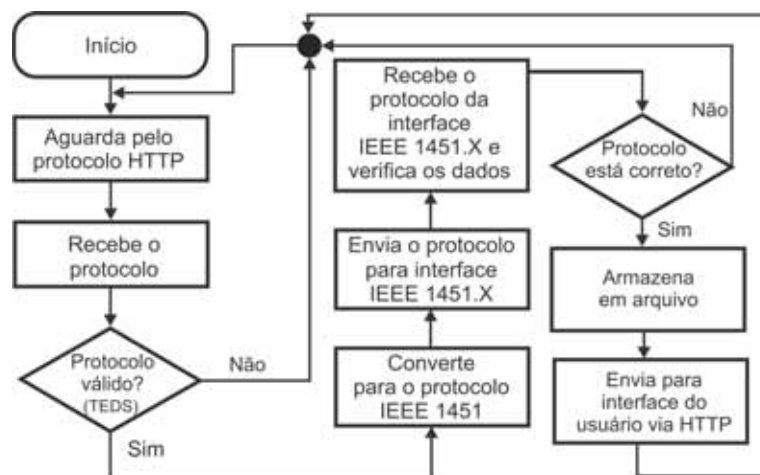


Fonte: Santos Filho (2012)

Na opção “TEDS” apresentam-se quais os TEDS que estão armazenadas no nó de rede NCAP. Neste trabalho, foram descritas as 4 tabelas obrigatórias para cada TIM, sendo que cada tabela TEDS foi armazenada em arquivos separados e especificada de acordo com a interface de comunicação a que pertence. Como exemplo de descrição do nome do arquivo, têm-se, RS232_Meta_TEDS, em que RS-232 define a interface e Meta-TEDS descreve que tabela TEDS está sendo descrita.

O NCAP reconhece o TIM quando conecta ambos e transfere o UUID do TIM para NCAP, caso o identificador não esteja presente no NCAP é feita a leitura dos TEDS e armazenado em arquivo separadamente. No entanto, se o identificador UUID encontra-se presente no NCAP, então é realizada a leitura dos TEDS no próprio NCAP, para que seja feito o reconhecimento. Na Figura 93 apresenta-se o fluxograma de leitura dos TEDS.

Figura 93 – Fluxograma da leitura dos TEDS.



Fonte: Santos Filho (2012)

Para visualizar os TEDS define-se primeiramente a interface em que se deseja realizar a leitura, em seguida, o módulo. Na Figura 94 apresenta-se a interface usuário de seleção dos módulos de leitura dos TEDS.

Figura 94 – Interface de seleção da leitura dos TEDS.

TEDS's Description

Define interface:

Define module:

Módulo_1

Módulo_2

Fonte: Santos Filho (2012)

Na Figura 95 apresenta-se um exemplo de leitura dos TEDS.

Para realizar a leitura dos sensores e controle dos atuadores é utilizada a opção “Transducer Access”. Ao acessar a opção “Transducer Access” são disponibilizadas para o usuário opções de definição da interface de comunicação (ZigBee/RS-232).

Figura 95 – Exemplo de leitura dos TEDS armazenadas em arquivo no NCAP.

Meta TEDS

Field Type	Field Name	Description	Data Type	Lenght	Value
---	Lenght	Octets Numbers	UInt32	4	0 0 0 25
3	TEDSID	Identification TEDS	UInt32	4	0 1 1 1
4	UUID	Universal Unique Identification	UUID	10	8 FB 61 B4 80 81 F6 43 A1 B1
A	OHoldOff	Time Limit to response	Float32	4	40 A0 0 0
C	TestTime	Time to self test	Float32	4	40 0 0 0
D	MaxChan	Transducer channel number	UInt16	2	0 3
----	Checksum	----	UInt16	2	6 2F

Transducer Channel TEDS - Channel 1

Field Type	Field Name	Description	Data Type	Lenght	Value
---	Lenght	Num. de octetos na META TEDS	UInt32	4	0 0 0 53

Fonte: Santos Filho (2012)

Ao definir a interface de comunicação, o nó de rede realiza a leitura dos TEDS referente ao TIM que esteja conectado e apresenta as características do módulo de acordo com a tabela TEDS definida e quais os transdutores que fazem parte do TIM. Na Figura 96 apresenta-se exemplo das características do TIM referente ao módulo STIM1-RS232 implementado em laboratório.

Na Figura 97 apresentam-se os campos “ChannelID”, “XdcrOperate”, “ReadData” e “Lenght of Command” referente ao protocolo baseado no padrão IEEE 1451.0. No Apêndice B, apresentam-se exemplos de comandos para requisição da temperatura e controle dos atuadores.

Ao preencher os campos e submeter os dados, o microcomputador envia o protocolo IEEE 1451.0 para o NCAP, encapsulando os dados no quadro Ethernet. Ao receber os dados, o NCAP obtém o protocolo IEEE 1451.0 do quadro Ethernet, verifica a integridade dos dados e envia para uma de suas interfaces de comunicação definida pela norma IEEE 1451. O TIM, ao receber o protocolo IEEE 1451.0, realiza a verificação da classe do comando, função e o canal do transdutor. Para os atuadores é analisado o conteúdo do campo de dados contendo informações de controle. É importante ressaltar que ambas interfaces com fio e sem fio, neste trabalho, realiza a comunicação ponta a ponto e os protocolos utilizados são semelhantes para ambas interfaces. Na Figura 98 apresenta-se o fluxograma do nó de rede NCAP para acesso aos transdutores inteligentes.

Para realizar a requisição da temperatura, foi utilizado o comando: 00_H 02_H 03_H 01_H

Figura 96 – Descrição dos TEDS referente ao módulo STIM1_RS232.

Transducer Module	
Module Identification:	8 FB 61 B4 80 81 F6 43 A1 B1
Time Limit to response	5.0000
Time to self test	2.0000
Transducer channel number	3
Transducer - Channel 1	
Calibration Key	0
Channel Type	Sensor
Physical Units	32 1 0 39 1 82
Low Limit	0.0000
High Limit	55.0000
Operational Error	0.5000
Self Test	1
Sample	28 1 0 29 1 1 30 1 8
TransducerChannel update time	0.1000
TransducerChannel read setup time	0.0000

Fonte: Santos Filho (2012)

Figura 97 – Campos para controle ou monitoramento dos transdutores conectados ao TIM.

TransducerChannel self-test time requirement	30.0000
Sampling	31 1 0
End-of-data-set operation attribute	1
Data Transmission Attribute	1
Actuator-halt attribute	1
Module Name:	S T I M 1 _ R S 2 3 2

Monitoring and Control - RS232

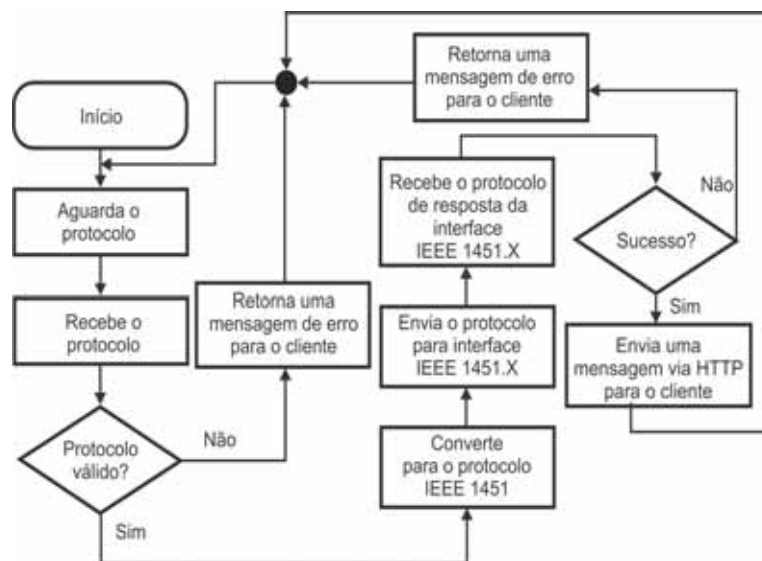
Tranducer Channel:	
Destination Transducer	<input type="text"/> ChannelID
Command Class	<input type="text"/> XdcrOperate
Command Function	<input type="text"/> ReadData
Lenght	<input type="text"/> Lenght of Command
Command	<input type="text"/>
Command	<input type="text"/>
<input type="button" value="Submit"/>	

Fonte: Santos Filho (2012)

00_H 06_H 00_H 00_H 00_H 00_H 00_H 00_H. A forma de onda referente ao protocolo de requisição é apresentada na Figura 99. Na Figura 100 apresenta-se a forma de onda referente ao comando: 01_H 00_H 06_H 00_H 00_H 00_H 00_H 00_H 6D_H 0D_H 0A_H correspondente à resposta do STIM1_RS232 para o NCAP.

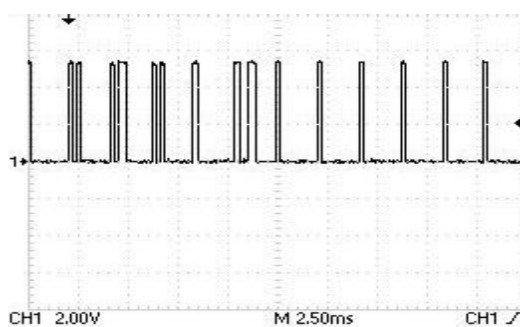
Na Figura 101 apresenta-se resposta do canal do sensor de temperatura, utilizando a

Figura 98 – Fluxograma do *software* de acesso aos transdutores inteligentes.



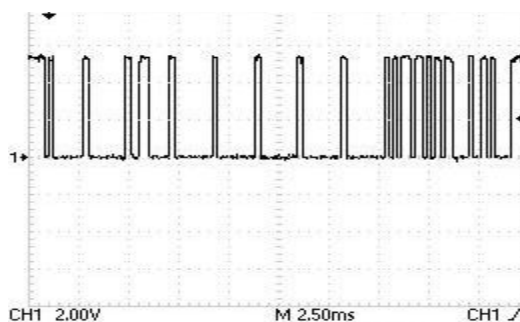
Fonte: Santos Filho (2012)

Figura 99 – Forma de onda do comando enviado do NCAP para o STIM1_RS232.



Fonte: Santos Filho (2012)

Figura 100 – Forma de onda do comando enviado do STIM1_RS232 para o NCAP.



Fonte: Santos Filho (2012)

interface de comunicação RS-232.

Para o controle do motor de passo, foi utilizado o comando: $00_H 02_H 03_H 02_H 00_H 06_H 00_H 00_H 00_H 00_H 01_H 0A_H$. Na Figura 102 apresenta-se o comando de requisição do

Figura 101 – Resposta do módulo STIM1_RS232 referente a temperatura.

Transdutores

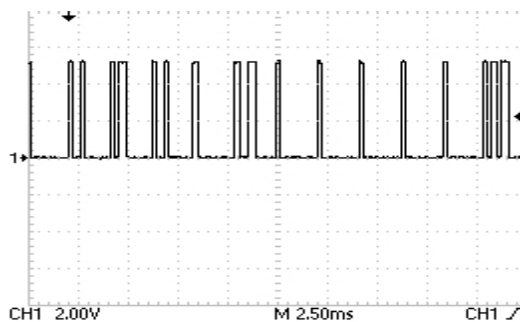
Sensores

Canal 1: 22 °C

Fonte: Santos Filho (2012)

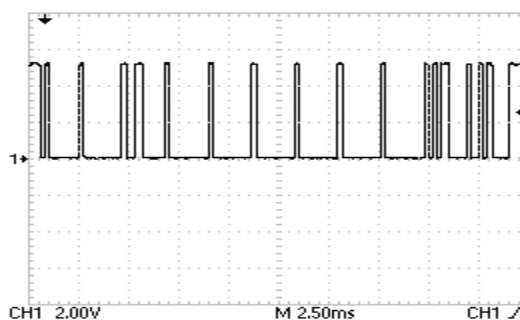
NCAP para STIM1_RS232 em forma de onda e, na Figura 103 o comando de resposta do STIM1_RS232 para o NCAP.

Figura 102 – Forma de onda do comando enviado do NCAP para o STIM1_RS232.



Fonte: Santos Filho (2012)

Figura 103 – Forma de onda do comando enviado do STIM1_RS232 para o NCAP.



Fonte: Santos Filho (2012)

Na Figura 104 apresenta-se a interface usuário com quantos graus o motor de passo foi movimentado.

Figura 104 – Resposta do módulo STIM1_RS232 referente ao motor de passo.

Transdutores

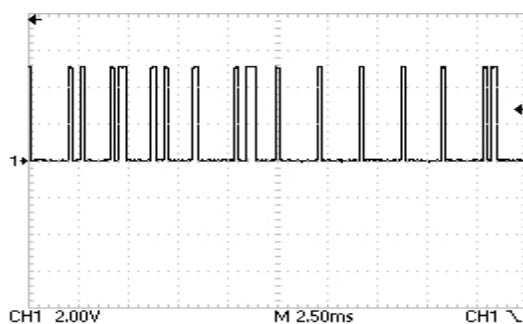
Atuadores

Canal 2: 10

Fonte: Santos Filho (2012)

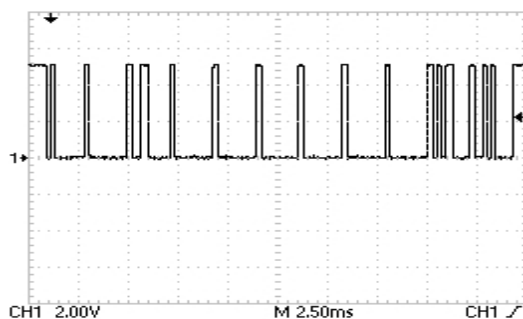
Para o acionamento do motor CC implementado no módulo WTIM1_ZigBee, foi utilizado o comando: $00_H 02_H 03_H 02_H 00_H 06_H 00_H 00_H 00_H 00_H 01_H$. Na Figura 105 apresenta-se o comando de requisição do NCAP para WTIM1_ZigBee em forma de onda e na Figura 106, o comando de resposta do WTIM1_ZigBee para o NCAP.

Figura 105 – Forma de onda do comando enviado do NCAP para o WTIM1_ZigBee.



Fonte: Santos Filho (2012)

Figura 106 – Forma de onda do comando enviado do WTIM1_ZigBee para o NCAP.



Fonte: Santos Filho (2012)

Na Figura 107 apresenta-se a interface usuário representando o estado do atuador, o valor 1 representa ligado.

Figura 107 – Resposta do módulo WTIM1_ZigBee referente ao controle do motor CC.

Transdutores

Atuadores

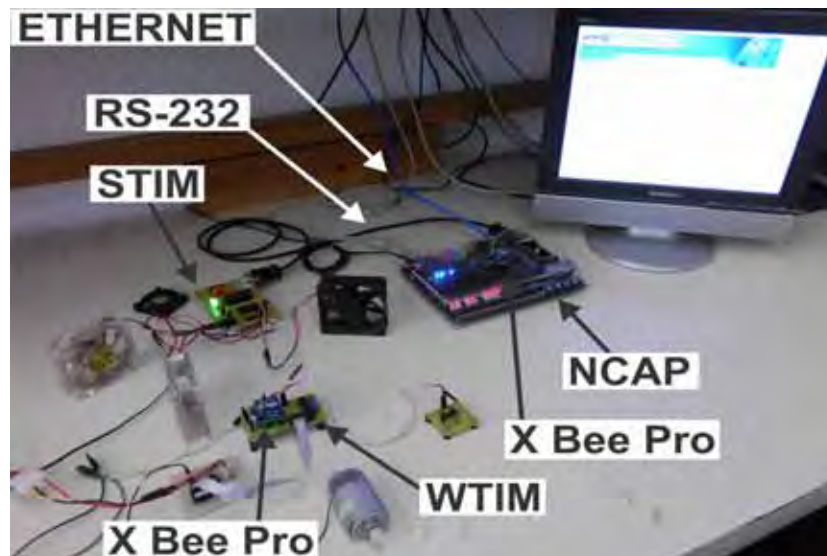
Canal 2: 1

Fonte: Santos Filho (2012)

A leitura da temperatura para os módulos WTIM2_ZigBee e WTIM1_ZigBee são realizados, utilizando os mesmos comandos referentes ao módulo STIM1_RS232, porém, com canais referentes a cada sensor. Nos Apêndices B.3 e B.4 apresentam-se os comandos para leitura dos sensores nos WTIMs. Para o módulo STIM2_RS232, o comando para desligar ou acionar os ventiladores podem ser visualizados no Apêndice B.2.

Na Figura 108 apresentam-se a foto do sistema desenvolvido em laboratório, o nó de rede IEEE 1451.1 e as duas interfaces de comunicação com os TIMs.

Figura 108 – Foto do sistema desenvolvido em laboratório.



Fonte: Santos Filho (2012)

Capítulo 7

Conclusões Gerais

7.1 Conclusões

Diferente de outras aplicações baseadas em microcontroladores, ASICs e outras soluções comerciais, o uso da linguagem de descrição de *hardware* e a atual tecnologia de programação permitem desenvolver sistemas com todas as capacidades e com a flexibilidade dos processadores embarcados e funções personalizadas. Esta abordagem permite a introdução de aspectos relevantes na criação de sistemas digitais, tais como a reutilização de código. Essa flexibilidade torna possível a criação de *frameworks* baseada no IEEE 1451 muito útil para IEEE 1451.X e aplicações portáteis.

No presente trabalho, apresentou-se o desenvolvimento de um nó de rede utilizando tecnologia das FPGAs e o sistema operacional embarcado μ Clinux, capaz de receber dados de uma rede externa (Ethernet), processar os dados convertendo para um padrão IEEE 1451.0-2007 e enviar através de uma de suas interfaces de rede interna baseadas na norma IEEE P1451.2 ou IEEE 1451.5. Uma das principais características do nó de rede desenvolvido é ser totalmente embarcado e autônomo, realizando o roteamento de cada pacote, baseado em uma de suas normas do padrão IEEE 1451. Além de ser embarcado, o *hardware* e o sistema operacional embarcado foram desenvolvidos baseados nas características da aplicação, minimizando os custos em relação a dispositivos e processamento de recursos não utilizados na aplicação. Tais características foram validadas no trabalho, sendo que, para o desenvolvimento do nó de rede, foram utilizados somente os periféricos necessários para o seu desenvolvimento, como, por exemplo, a interface SPI, para comunicação com o *SD card*, 3 interfaces UART, sendo uma para interface sem fio e uma para interface cabeada.

A implementação de um sistema operacional embarcado em um dispositivo dedicado

facilita o desenvolvimento de outros projetos, pois oferece recursos pré-definidos, como, por exemplo o servidor FTP, servidor web Boa, editor de texto vi para configurações do sistema, comandos básicos do Linux focados para os microcomputadores, acesso remoto, *driver* para rede Ethernet, *driver* para o *SD Card* e acesso à interface RS-232.

Um outro aspecto importante diz respeito à implementação do SD Card para armazenamento dos TEDS em arquivo sendo que o sistema operacional μ Clinux suporta no máximo 2 GB. No entanto, é quantidade suficiente para o armazenamento de grande quantidades de dados de descrição dos TEDS.

A escolha da rede Ethernet para acesso ao nó de rede NCAP justifica-se pelo fato de possibilitar a construção de uma rede de baixo custo, difundida em várias áreas e ser padronizada pela IEEE 802.3. Uma outra característica que deve ser ressaltada é a configuração da rede Ethernet que, através do modo de comando do μ Clinux, é possível aplicar em outras redes Ethernet, realizando configurações básicas, como modificação do endereçamento IP, *gateway* e máscara.

Uma característica que deve ser ressaltada e que foi diferenciada das demais aplicações foi o desenvolvimento do *software* do NCAP utilizando linguagem de programação estruturada, apesar de vários autores trabalharem com linguagens Orientada a Objeto. A norma sugere que não é obrigatório o desenvolvimento do sistemas utilizando conceitos de orientação a objetos. Neste trabalho, utilizou-se a linguagem C, que oferece suporte ao acesso aos periféricos e às páginas web utilizando conceitos de CGI.

A característica de leitura dos sensores através do navegador foi possível devido ao fato de que o sistema operacional embarcado possui um servidor web para armazenamento de páginas em dispositivos embarcados, característica que facilita o acesso aos dados dos sensores e o controle dos atuadores através da rede Ethernet. Tais características facilitam o monitoramento e o controle dos transdutores inteligentes conectados ao nó de rede, sendo possível realizar o acesso através de diferentes navegadores webs, necessitando que o usuário esteja conectado à mesma rede Ethernet em que o nó de rede se encontra.

Para realização dos testes, foram feitos, em laboratório quatro TMS contendo sensores e atuadores. As requisições de monitoramento e controle foram realizadas com sucesso, validando o resultado final do projeto.

7.2 Contribuições

Uma contribuição importante do trabalho foi a implementação do sistema operacional embarcado em uma arquitetura dinâmica, fornecendo uma flexibilidade ao sistema, ampliando a área de pesquisa e apresentando uma nova metodologia de desenvolvimento do padrão IEEE 1451. Esta pode ser considerada a contribuição global do trabalho, entretanto, há uma série de contribuições específicas tais como:

1. O desenvolvimento de um nó de rede com duas interfaces aumenta a flexibilidade do sistema ampliando as áreas de aplicação;
2. A forma de implementação dos TEDS em arquivo no nó de rede facilita a implementação de novos TIMS, pois podem ser obtidas através do TIM ou descrita no próprio NCAP;
3. O desenvolvimento de uma nova metodologia de desenvolvimento do *software* NCAP utilizando programação estruturada;
4. O desenvolvimento de uma arquitetura e um sistema operacional embarcado dinâmico focada para redes de transdutores inteligentes, utilizando a norma IEEE 1451;
5. O desenvolvimento de TIMs de baixo custo utilizando microcontroladores da Atmel;
6. Utilização da norma IEEE 1451.0-2007 para a descrição dos TEDS, utilização da norma IEEE 1451.1 para o desenvolvimento do NCAP e da norma IEEE P1451.2 e IEEE 1451.5 para descrição dos TIMs. A implementação do nó de rede baseado no padrão IEEE 1451 apresenta os seguintes benefícios: o desenvolvimento de um modelo de sistema uniforme, modelos de aplicações portáteis, interoperabilidade de comunicação, um modelo de informação uniforme para representar parâmetros físicos de dados e característica de conexão *plug and play* entre o NCAP e o TIM.

7.3 Sugestões de Trabalhos Futuros

Como contribuição para trabalhos futuros, propõem-se os seguintes:

- Implementação da interface *multidrop* baseada na especificação IEEE 1451.3;
- Implementação da interface MMI entre o nó de rede NCAP e TIM;

- Implementação da rede CAN baseada no padrão IEEE P1451.6;
- Implementação da interface RFID entre o nó de rede NCAP e TIM baseada na especificação IEEE 1451.7;
- Implementação da JVM (*Java Virtual Machine*), para o desenvolvimento do nó de rede NCAP e realização de análise de desempenho entre *software* estruturado e o orientado a objeto em sistemas embarcados;
- Desenvolvimento de dois ou mais processadores no kit DE2, para análise do desempenho do nó de rede IEEE 1451, utilizando processamento paralelo com múltiplas interfaces;
- Implementação de um *clock* externo e configuração do *hardware* e do sistema operacional embarcado para comunicação com dispositivos USB;
- Desenvolvimento de um *software* para descrição dos TEDS no nó de rede NCAP;
- Desenvolvimento de uma rede *mesh* utilizando módulos XBee Series 2 e implementação da interface sem fio baseado no padrão IEEE 802.11;
- Instalação e configuração do banco de dados MySQL no nó de rede NCAP para realizar o armazenamento dos dados obtidos dos transdutores inteligentes e dos TEDS;
- Implementação do TIM no kit DE1 utilizando tecnologia de *hardware* dinâmico em FPGA e sistema operacional embarcado;
- Instalação e configuração do sistema operacional embarcado RTOS (*Real Time Operate System*) para análise de desempenho e comparação com o sistema operacional μ Linux.

Referências

- ALTERA. Introduction to the altera SoPC builder. In: _____. **Integration of the nios II system into a quartus II project**. [S.l.: s.n.], 2006. p. 19. Disponível em: <<http://www.altera.com/literature/ds/dsnios2perf.pdf>>. Acesso em: 20 maio 2008.
- ALTERA. Altera development and education boards. In: _____. **Nios II performance benchmarks datasheet. Nios II performance benchmarks**. [S.l.: s.n.], 2008. Disponível em: <<http://www.altera.com/education/univ/materials/boards/unv-dev-edu-boards.html>>. Acesso em: 20 jul. 2008.
- ALTERA. Nios II performance benchmarks. In: _____. **Nios II performance benchmarks datasheet**. [S.l.:s.n.], 2009. p. 6. Disponível em : <<http://www.altera.com/literature/ds/dsnios2perf.pdf>>. Acesso em: 21 maio 2011.
- ATMEL. **8 bit AVR with 8KBytes in system programmable flash**. [S.l.: s.n.], 2011. 302 p.
- AXELSON, J. **Serial port complete - COM ports, USB virtual COM ports, and ports for embedded systems**. 2. ed. [S.l.: s.n.], 2007. 400 p.
- BODSON, D. IEEE vehicular technology. **IEEE Vehicular Technology Magazine**, New York, v. 7, n. 1 , p. 66-69, 2010.
- BUENO, M. A. F.; BRASIL, C. R. S.; MARQUES, E. Sistema operacional embarcado eCos com suporte a SMP para o processador nios II. In: CONGRESSO DA SBC WSO, 27.; WORKSHOP DE SISTEMAS OPERACIONAIS, 4., 2007, Rio de Janeiro. **Anais...** Rio de Janeiro: [s.n.], 2007. p. 764-776.
- CANZIAN, E. **Comunicação serial - RS232**. Cotia: CNZ, 2008. p. 18. Disponível em: <<http://pt.scribd.com/doc/71511554/rs232>>. Acesso em: 20 Abr. 2010.
- CHENG, H.; QIN, H. A design of IEEE 1451.2 compliant smart sensor based on the nios soft-core processor. In : IEEE INTERNATIONAL CONFERENCE ON VEHICULAR ELECTRONICS AND SAFETY, 2005, China. **Conference....**, China: [s.n.], 2005. p. 193-198.
- COSTA, C. da. **Projetando controladores digitais com FPGA**. [S.l.]: NOVATEC, 2005. p. 159.
- DIONNE, D. J.; DURRANT, M. uClinux embedded linux/microcontroller project. In: TRADEMARKS OF ARCTURUS NETWORKS. **The embedded linux/microcontroller**

project is a port of Linux to systems without a memory management unit (MMU). [S.l.: s.n.], 2002. Disponível em: <<http://www.uclinux.org/description/>>. Acesso em: 20 fev. 2008.

ECCLES, L. H. The need for smart transducers - an aerospace test and evaluation perspective. **IEEE Instrumentation and Measurement Magazine**, Renton, v. 11, p. 23-28, 2008.

ECOS, A. **About eCos overview.** [S.l.: s.n.], 2002. Disponível em: <<http://ecos.sourceforge.org/about.html>>. Acesso em: 21 de fev. 2008.

FARAHANI, S. **Zigbee wireless networks and transceivers.** [S.l.: s.n.], 2008. 364 p.

GUNDAVARAM, S. **CGI programming on the world wide web.** [S.l.]: O Reilly Associates, 1996. 433 p.

HIGUERA, J.; POLO, J.; GASULLA, M. A Zigbee wireless sensor network compliant with the IEEE 1451 standard. In: IEEE SENSORS APPLICATIONS SYMPOSIUM, 2009, Barcelona. **Symposium...** Barcelona: Univ. Politec. de Catalunya, 2009. p. 309-313

IEEE. **IEEE standard for a smart transducer interface for sensor and actuators - network capable application processor (NCAP) information model.** [S.l.]: IEEE Instrumentation and Measurement Society, Sponsored by the Technical Committee on Sensor Technology, 1999. 341 p.

IEEE. **IEEE standard for a smart transducer interface for sensors and actuators mixed mode communication protocols and transducer electronic data sheet (TEDS) formats.** [S.l.]: IEEE Instrumentation and Measurement Society, Sponsored by the Technical Committee on Sensor Technology, 2004. 439 p.

IEEE. **Draft standard for a smart transducer interface for sensor and actuators a high speed CANopen based transducer network interface for intrinsically safe and non-intrinsically safe applications.** [S.l.: s.n.], 2007. Disponível em: <<http://grouper.ieee.org/groups/1451/6/index.htm>>. Acesso em: 20 abr. 2011. The Proposed IEEE 1451.6 Standard

IEEE. **IEEE standard for a smart transducer interface for sensors and actuators - wireless communication protocols and transducer electronic data sheet (TEDS) formats.** [S.l.]: IEEE Instrumentation and Measurement Society, 2007. 246 p.

IEEE. **IEEE standard for a smart transducer interface for sensors and actuators - common functions, communication protocols, and transducers electronic Data Sheet (TEDS) formats.** [S.l.]: IEEE Instrumentation and Measurement Society, Sponsored by the Technical Committee on Sensor Technology, 2007. 323 p.

IEEE. **IEEE standard for a smart transducer interface dos sensors and actuators transducers to radio frequency identification (RFID) systems communication protocols and transducer electronic data sheet formats.** [S.l.]: IEEE Instrumentation and Measurement Society, Sponsored by the Technical Committee on Sensor Technology, 2010. 99 p.

JONES, M. T. **Busybox simples embedded linux systems - A small toolkit for small environments.** [S.l.]: Consultant Engineer, Emulex, 2006. 8 p.

- JUNQUEIRA, R. S. **PDA baseado no microprocessador nios II com sistema operacional uclinux embarcado em FPGA.** [S.l.]: Centro Universitário Positivo, Núcleo de Ciências Exatas e Tecnológicas, Engenharia da Computação, 2007. 49 p.
- KOBAYASHI, C. Y. **A tecnologia bluetooth e as aplicações.** [S.l.]: BCC - IME - Universidade de São Paulo, 2004. 5 p.
- KUROSE, J. F.; ROSS, K. W. **Redes de computadores e a internet.** 3. ed. São Paulo: Person, 2005. 656 p.
- LARRAURI, J. et al. A bluetooth sensor network based on the IEEE 1451 standard - A sensor network solution to evaluate the wellbeing of the passenger and improve safety in cars. In: INTERNATIONAL CONFERENCE ON, 2010, [S.l.]. **Proceedings...** [S.l.]: Wireless Information Networks and Systems WINSYS, , 2010. 1-6 p.
- LEE, K.; SONG, E. Object - oriented application framework for IEEE 1451.1 standard. **Instrumentation and Measurement, IEEE Transactions on**, New York, v. 54, n. 4, p. 1527 - 1533, 2005.
- LEE, K.; SONG, E. Wireless sensor network based on IEEE 1451.0 and IEEE 1451.5-802.11. In: THE EIGHTH INTERNATIONAL CONFERENCE ON ELECTRONIC MEASUREMENT AND INSTRUMENTS, 2007, Gaithersburg, 2007. **Conference...** Gaithersburg: [s.n.], 2007. p. 5.
- LEE, K.; SONG, E. Y. **UML model for the IEEE 1451.1 standard.** [S.l.]: Instrumentation and Measurement Technology Conference- IMTC , 2003. 1687-1591 p.
- LEE, K. B. **P1451.3 working group.** [S.l.]: National Institute of Standards and Technology, 2009. Disponível em: <<http://www.nist.gov/el/isd/ieee/1451group3.cfm>>. Acesso em: 10 mar. 2011.
- LIMA, F. S. de. **Estratégia de escalonamento de controladores PID baseadas em regras Fuzzy para as redes industriais foundation fieldbus usando blocos padrões.** 2004. 57 f. Dissertação (Mestrado)- Centro de Tecnologia, Universidade Federal do Rio Grande do Norte, Rio Grande do Norte , 2004.
- LUTZ, P.; FAERBER, H. M. **Device drivers and test application for a SOPC solution with nios II software processor and uclinux.** Augsburg: University of Applied Sciences, 2009. 56 p.
- MANDA, S.; GURKAN, D. **IEEE 1451.0 compatible TEDS creation using .Net framework.** [S.l.]: Sensors Applications Symposium, 2009. 6 p.
- NEMETH-JOHANNES, J.; SWEETSER, V.; SWEETSER, D. **Implementation of an IEEE 1451.0-1451.5 compliant wireless sensor module.** Baltimore: Autotestcon, 2007. 364 - 371 p.
- NIKKANEN, K. **uClinux as an embedded solution.** [S.l.]: Turku Polytechnic Telecommunications Embedded Systems, 2003. 72 p.
- NIST. **P1451.3 working group.** [S.l.]: Department of Commerce: National Institute of Standards and Technology- NIST, 2011. Disponível em: <<http://www.nist.gov/el/isd/ieee/1451group3.cfm>>. Acesso em: 20 Abr. 2010.

- O'REILLY, T. et al. **Instrument interface standards for interoperable ocean sensor networks**. Monterey: Monterey Bay Aquarium Res. Inst, 2009. p. 10.
- PEREIRA, F. **Microcontroladores PIC técnicas avançadas, baseada no PIC 16F627 e 16F628**. 5. ed. São Paulo: [s.n.], 2007.
- PYMBLE, S.; **XBEE-PRO - Embedded Serial Zigbee Module PRO Version**, [S. l.]: Pymble Software. 2011, Disponível em: <<http://pymblesoftware.com/store/index.php/xbee-pro-embedded-serial-zigbee-module-pro-version.html>>. Acesso em: 21 Abr. 2011.
- RAMOS, H. G. **IEEE standard 1451 and a proposed time synchronization approach**. Lisbon: IEEE Instrumentation and Measurement Magazine, 2008. p. 29-37.
- REILLY, T. O. et al. **Excerpts from instrument interface standard for interoperable ocean sensor networks**. Monterey: ESONEWS, 2010. 5 p.
- SANTOS FILHO, T. A. dos. **Desenvolvimento de um nó o de rede com diferentes interfaces de acordo com o padrão IEEE 1451 utilizando o processador nios II e o sistema operacional embarcado uClinux**. 2012. 183 f. Tese (Doutorado)- Faculdade de Engenharia, Universidade Paulista "Júlio de Mesquita Filho", Ilha Solteira, 2012.
- SCHILDT, H. **C completo e total**. [S.l.]: Makron Books, 1997. 410 p.
- SEMICONDUCTOR, F. **Block guide - serial peripheral interface (SPIV3) block description**. [S.l.: s.n.], 2011. 22 p.
- SIGLA, A.; MORRIS, J. **Design of a test suite for NCAP-to-NCAP communication based on IEEE 1451**. Houston: [s.n.], 2008. 5 p.
- SONG, E.; LEE, K. Understanding IEEE 1451-networked smart transducer interface standard - what is a smart transducer? **Instrumentation e Measurement Magazine**, New York, v. 11, n. 2, , p. 11-17, 2008.
- SONG, E. Y.; LEE, K. B. Sensor network based on IEEE 1451.0 and IEEE p1451.2-RS232. In: IEEE INTERNATIONAL INSTRUMENTATION AND MEASUREMENT TECHNOLOGY CONFERENCE- I2MTC, 2008, Victoria. Conference... Victoria: [s.n.], 2008. 6 p.
- SONG, E. Y.; LEE, K. B. Interoperability test of IEEE 1451.5 standard based wireless sensors. In: INTERNATIONAL CONFERENCE ON MEASURING TECHNOLOGY AND MECHATRONICS AUTOMATION, 2010, Washington. Conference... Washington: [s.n.], 2010. p. 510-515.
- SONG, E. Y. et al. An IEEE 1451.5 - 802.11 standards based wireless sensor network with embedded WTIM. In: IEEE INTERNATIONAL INSTRUMENTATION AND MEASUREMENT TECHNOLOGY CONFERENCE, 2011, Gaithersburg. **Conference...** Gaithersburg: [s.n.], 2011. p. 1201-1206.
- STADZISZ, P. C. **Microsoft windows XP embedded**. [S.l.]: Centro Federal de Educação Tecnológica do Paraná - CEFET, 2003. Disponível em: <<http://www.ctse.citec.ct.utfpr.edu.br/contact.htm>>. Acesso em: 10 mar. 2008.

- STEPANENKO, A. et al. **Development of a minimal IEEE 1451.1 model for microcontroller implementation**. Houston: IEEE Sensors Applications Symposium, 2006. 6 p.
- SYSTEMS, J. S. **C Executive r and PSX(tm) real-time kernels**. [S.l.:s.n.]: 1998. Disponível em: <<http://www.jmi.com/cexec.html>>. Acesso em: 15 mar. 2008.
- TANENBAUM, A. S.; WETHERALL, D. **Redes de computadores**. 5. ed. [S.l.]: Person, 2011. 582 p.
- TECHNOLOGY. **Case study verilog vs VHDL**. [S.l.]: strategic computing and communications. [S.l.: s.n.], 2011. Disponível em: <<http://www.fpgarelated.com/showarticle/19.php>>. Acesso em: 4 out. 2011.
- TERASIC. **Development and education board getting started guide**. In: ALTERA. [S.l.: s.n.], 2005. p. 85. Disponível em: <<http://users.ece.gatech.edu/hamblen/DE2/DE2ReferenceManual.pdf>>. Acesso em: 20 maio 2008.
- THORNE, N. **A uClinux driver system for the nios2 processor**. Rondebosch : University of Cape Town, 2007. 45 p. Altera Wiki: Department of Electrical Engineering.
- TU, J. F. Create a TII of IEEE 1451.1 under UART. Proceeding of the Eighth In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING AND CYBERNETICS, 8., 2009, Taipei. **Proceeding ...** Taipei; [s.n.], 2009.p. 3324-3328.
- VIEGAS, V.; PEREIRA, M.; GIRAO, P. A brief tutorial on the IEEE 1451.1 standard. **IEEE Instrumentation and Measurement**, New York, , v. 13, n. 2, p. 38-45, 2008.
- WANG, Y. et al. A smart thermal environment monitor based on IEEE 1451.2 standard for global networking. **Instrumentation and Measurement, IEEE Transactions on**, New York, p. 1321 - 1326, 2005. ISSN 0018-9456.
- WENTWORTH, S. Boa: an embedded web server. Sid takes a look at Boa, a web server suitable for embedded systems. **Linux Journal**, [S.l.], 2001. Disponível em: <<http://www.linuxjournal.com/article/4773>>. Acesso em: 21 dez. 2009.
- WIEDENHOFT, R. G.; HOELLER, A.; FROHLICH, A. A. **Um gerente de energia para sistemas profundamente embarcados**. In: CONGRESSO DA SBC WSO, 17.; WORKSHOP DE SISTEMAS OPERACIONAIS, 4., 2007, Rio de Janeiro. **Anais...** Rio de Janeiro: [s.n.], 2007. p. 796-804.
- WOBSCHALL, D. Network sensor monitoring using the universal IEEE 1451 standard. **IEEE Instrumentation and Measurement Magazine**, New York, v 11, n. 2, p. 18-22, 2008.
- WOBSCHALL, D.; MUPPARAJU, S. **Low power wireless sensor with SNAP and IEEE 1451 protocol**. Atlanta: IEEE Sensors Applications Symposium, 2008. p. 225-227.

Apêndice A

μ Clinux-dist

A.1 Instalação do μ Clinux-dist

Para a instalação do ambiente de criação da imagem do μ Clinux deve-se instalar o sistema operacional Linux no microcomputador e acessar o sistema como superusuário denominado no ambiente Linux como root. Para a instalação do ambiente, neste trabalho foi utilizado o sistema operacional Fedora, uma distribuição Linux disponível gratuitamente na internet. Para a instalação dos pacotes do μ Clinux-dist, é necessário realizar diversos comandos utilizando terminal **shell** como usuário root. Para representar os comandos do ambiente linux, neste trabalho, os comandos serão precedidos pelo símbolo **#** (cerquilha). O primeiro comando é utilizado para atualizar o sistema Fedora com as bibliotecas necessárias para a instalação do ambiente μ Clinux-dist, descrito como:

```
#sudo yum install git-all git-gui make gcc ncurses-devel bison  
byacc flex gawk gettext ccache zlib-devel gtk2-devel lzo-devel  
pax-utilslibglade2-devel
```

Para armazenamento dos dados, foi utilizado um **hard disk** de 20 GB, no qual, o espaço mínimo requerido para instalação do sistema é de 6 GB. Para baixar o arquivo **nios2-linux-20090929.tar** da internet em modo de comando, foi utilizada a ferramenta **wget**, contendo os seguintes parâmetros:

```
#wget http://www.niosftp.com/pub/linux/nios2-linux-20090929.tar
```

Para verificação dos dados corrompidos é utilizado o comando:

```
#sha1sum nios2-linux-20090730.tar 1d99a54d36759cc6ce5f054ff0460  
b1bd370b0b6 nios2-linux-20090929.tar
```

Para descompactar o arquivo **nios2-linux-20090730.tar**, foi utilizada, a ferramenta **tar**. O arquivo pode ser descompactado em qualquer diretório, porém, para controle dos arquivos do sistema, neste trabalho, foi descompactado dentro do diretório **#/opt/**, de acordo com o comando:

```
#tar xf <caminho do arquivo>nios2-linux-20090929.tar
```

Ao descompactar o arquivo é realizada uma verificação do conteúdo dentro do diretório **nios2-linux**, através dos comandos:

```
#cd nios2-linux
```

```
#ls
```

O diretório **nios2-linux** deve possuir os arquivos e diretórios: **binutils**, **insight**, **u-boot**, **use_http_for_update**, **checkout**, **linux-2.6**, **uClibc**, **use_ssh443_for_update**, **elf2flt**, **README**, **uClinux-dist**, **gcc3**, **sshkey**, **update**, **glibc**, **toolchain-build** e **use_git_for_update**. Para verificação do sistema é utilizado o comando:

```
#./check
```

Ao executar o comando **./check**, o μ Clinux Nios2 e gnutools possuem os seguintes diretórios:

- **linux-2.6**: código fonte do kernel Linux, que é o Linux kernel padrão mais caminhos específicos do Nios2. A versão atual é v2.6.30.
- **uClinux-dist**: bibliotecas do μ Clinux e aplicativos. Local de construção do μ Clinux.
- **binutils**, **gcc3**, **elf2flt**, **insight** : o gnu fonte de ferramentas.
- **uClibc** : principais bibliotecas, é menor que a glibc.
- **U-boot** : gerenciador de inicialização e monitor, DasUBoot .

Para os microcomputadores de 64 bits, é necessário mudar o **arch** para 32 bits, utilizando o **setarch i386** através dos comandos:

```
#cd toolchain-build
```

```
#gcc -version
```

```
#git clean -f -x -d
```

```
#make gcc elf2flt gdb-host
```

```
#cd toolchain-build
```

Para utilizar o compilador gcc em qualquer diretório, foi necessário alterar o arquivo **/bash_profile**, adicionando no final o comando:

```
#PATH=$PATH:/home/hippo/nios2-linux/toolchain-build/build
/nios2/bin
```

A alteração foi finalizada realizando o **logout** e **login** no sistema operacional Linux, e a verificação foi feita compilando e executando um programa feito em linguagem C, através do comandos:

```
#nios2-linux-uclibc-gcc -v    #nios2-linux-uclibc-ftthdr hello
```

A.2 Configuração da imagem do μ Clinux

A configuração e a definição das aplicações do μ Clinux foram definidas utilizando a ferramenta menuconfig. O menuconfig é executado no diretório uClinux-dist, localizado neste trabalho em `/opt/nios2-linux/uClinux-dist/`, através do comando:

```
#make menuconfig
```

Ao executar comando menuconfig, são apresentadas ao projetista as opções de configuração: **Vendor/Product Selection** e **Kernel/Library/Default Slection**.

Para definir o processador em que o sistema operacional será executado, primeiramente defina-se o fabricante através do opção **Vendor/Product Selection** que apresentará duas opções:

```
Vendor/Product Selection —>
  — Select the Vendor you wish to target
  Vendor (Altera) —>
    — Select the Product you wish to target
    Altera Products (nios2) —>
```

Para definir os componentes do μ Clinux como **drivers**, interfaces, sistema de arquivo, comandos e aplicativos são selecionados através da opção **Defaults Selection** em:

```
Kernel/Library/Defaults Selection —>
```

Ao selecionar a opção **Default Selection** o projetista poderá definir os tipos de configurações do sistema operacional μ Clinux sendo:

```
— Kernel is linux-2.6.x
  Libc Version (None) —>
    [ ] Default all settings (lose changes)
    [ ] Customize Kernel Settings
    [ ] Customize Vendor/User Settings
```

Update Default Vendor Settings

A primeira opção **Default all settings**, define as configurações padrões do sistema operacional μ Clinux. A opção **Customize Kernel Settings**, define os drivers e interfaces de comunicação. A terceira opção **Customize Vendor/User Settings**, refere-se aos comandos, aplicativos e servidores, e a quarta opção **Update Default Vendor Settings** realiza a atualização do sistema.

A.2.1 Configuração das Interfaces UART no Ambiente uClinux-dist

No kernel menuconfig, para trabalhar com dispositivo de console, deve ser definida uma das duas opções de interface: **JTAG UART** ou **serial UART**. Neste trabalho, foi definida a jtag UART, devido a serial UART ser utilizada para comunicação com módulo STIM. Para definir a comunicação com o STIM, através do sistema operacional μ Clinux, foram definidas as opções no menuconfig, cujas as opções de configurações são definida pelo caminho:

Device Drivers —> **Character devices** —> **Serial drivers** —>

Ao definir o link Serial drivers, serão apresentadas ao projetista as configurações da porta serial com as opções:

```
[ ]8250/16550 and compatible serial support
    *** Non-8250 serial port support ***
[*]Altera JTAG UART support
[*]Altera JTAG UART console support
    [ ]Bypass output when no connection
[*] Altera UART support
    (3) Maximum number of Altera UART ports
    (4800) Default baudrate for Altera UART ports
[ ] Altera UART console support
```

Neste trabalho, foi configurada a quantidade de portas seriais utilizadas no projeto e a velocidade de transmissão dos dados. Para o uso do cabo USB, em modo de comando do μ Clinux, seleciona-se o comando Altera JTAG UART console suporte.

A.2.2 Configuração da Interface SPI no Ambiente uClinux-dist

A interface SPI foi utilizada neste projeto para realizar a comunicação com a memória flash externa (SD card). O SD card foi utilizado para o armazenamento dos arquivos do NCAP, dados de log e leitura dos dados dos transdutores. Para o sistema operacional realizar a comunicação com a memória flash, deve-se definir o **driver** e o arquivo de sistema como, por exemplo: vfat ou ext3. No menuconfig, define-se o driver através da opção:

Device Drivers —>

Ao selecionar a opção **Device Drivers**, define-se suporte ao **driver** através das opções:

[*]SPI support —>

[*]Altera SPI Controller

[*]MMC/SD card support —>

[*]MMC block device driver

[*]Use bounce buffer for simple hosts

[]SDIO UART/GPS class support

[]MMC host test driver

*** MMC/SD Host Controller Drivers***

[*]MMC host test driver

[]NIOS SD/SDIO/MMC Host

Para ativar o suporte ao arquivo de sistema, selecionam-se as seguintes opções:

File systems —>

[*]Ext3 journalling file system support

DOS/FAT/NT Filesystems —>

[*]VFAT (Windows-95) fs support

(437)Default codepage for FAT

(iso8859-1) Default iocharset for FAT

-*- Native language support —>

[*]Codepage 437 (United States, Canada)

[*]NLS ISO 8859-1 (Latin 1; Western European Languages)

No ambiente μ Clinux em modo de comando, para ativar a nova unidade é realizado o comando:

```
#mount -t vfat /dev/mmcblk0p1 /mnt
```

A nova unidade será montada no diretório /mnt e pode ser acessada através do comando:

```
#cd /mnt
```

A.2.3 Configuração do Servidor Web, FTP e Acesso Remoto Telnet

O FTP é um protocolo de transferência de arquivos entre cliente/servidor e o Telnet é um protocolo utilizado para acesso remoto. Neste projeto, o FTP foi usado para enviar arquivos para o nó de rede NCAP, como por exemplo, os TEDS. Para ativar o servidor FTP no sistema operacional μ Clinux são definidas as seguintes opções:

Network Applications —>

```
[*] ftpd
```

```
[*]inetd
```

Para ativar o acesso remoto por Telnet, é definida a seguinte opção:

Network Applications —>

```
[*] telnetd
```

Para ativar o servidor FTP e Telnet, é realizada o comando no modo de comando do μ Clinux:

```
#inetd &
```

Para incluir o servidor web Boa no projeto, é selecionada a opção:

Network Applications —>

```
[*] boa
```

Para ativar o servidor Boa, é realizado o comando no modo de comando do μ Clinux:

```
#boa &
```

Para ativar os serviços na inicialização do sistema operacional μ Clinux é necessário editar o arquivo “rc” que se encontra no diretório ../uClinux-dist/romfs/etc/rc inserindo as linhas de comando:

```
#boa &
```

```
#inetd &
```

O servidor FTP por padrão armazena seus dados dentro do diretório /home/ftp. Para realizar a mudança de diretório do servidor FTP, é necessário editar o arquivo passwd

que encontra dentro do diretório `/uClinux-dist/vendors/Altera/nios2`. Neste trabalho, foi definido o diretório `/mnt/htdocs/cgi-bin`, referente à unidade do SD Card.

As páginas no servidor Boa são armazenadas por padrão no diretório `/home/httpd/`. Para realizar a alteração do diretório de armazenamento dos dados, é necessário editar o arquivo “`boa.conf`” que se encontra no diretório `../uClinux-dist/romfs/etc/boa/`. No arquivo “`boa.conf`” a linha que corresponde “`DocumentRoot /home/httpd/`” é alterada para o diretório em que se deseja armazenar as páginas web. Neste trabalho, as páginas web foram armazenadas dentro da unidade SD Card, realizando a seguinte alteração, “`DocumentRoot /mnt/htdocs`”.

A.2.4 Compilação de arquivos em linguagem C para ambiente μ Clinux

Os arquivos em linguagem C são compilados para o ambiente μ Clinux, utilizando o comando:

```
#nios2-linux-uclibc-gcc <nome do arquivo.c> -o <nome arquivo de saída> -elf2flt
```

O parametro `-elf2flt` é utilizado para criar o arquivo executável em formato FLAT reconhecido pelo ambiente μ Clinux. Para que o μ Clinux reconheça como um comando, o arquivo gerado deve ser copiado para dentro do diretório “`../uClinux-dist/romfs/bin`” utilizando o comando `cp <origem> <destino>`. Para realizar a transferência através do FTP e realizar a execução da aplicação é utilizado o comando “`./<nome do arquivo>`”.

Apêndice **B**

Comandos de Leitura e Controle

Neste Apêndice, apresentam-se os comandos de leitura, controle dos transdutores inteligentes e erro de acordo com cada TIM desenvolvido baseado na norma IEEE 1451.0.

B.1 Comando para Monitoramento e Controle do Módulo STIM_1

O STIM_1 foram utilizados 3 transdutores com respectivos canais:

- **Canal 2** - sensor de temperatura LM35.
- **Canal 3** - Motor de passo 1.
- **Canal 4** - Motor de passo 2.

Para a leitura do sensor de temperatura, o NCAP envia para o STIM_1 o comando: $00_H 02_H 03_H 01_H 00_H 06_H 00_H 00_H 00_H 00_H 00_H$, sendo:

- $00_H 02_H$ - Canal do transdutor de destino.
- 03_H - Classe do comando (Estado de operação do transdutor).
- 01_H - Função do comando (Leitura do transdutor).
- $00_H 06_H$ - Número de octetos.
- $00_H 00_H 00_H 00_H 00_H 00_H$ - Off-set.

Como resposta ao comando de requisição da temperatura é enviado do STIM_1 para o NCAP o comando: $01_H 00_H 06_H 00_H 00_H 00_H 00_H 00_H 73_H$, sendo:

- **01_H** - Campo diferente de 0 - sucesso.
- **00_H 06_H** - Número de octetos no campo de dados.
- **00_H 00_H 00_H 00_H 00_H** - **Off-set**.
- **73_H** - Valor da temperatura.

Para o controle do motor de passo 1, o NCAP envia para o STIM_1 o comando: 00_H 03_H 03_H 02_H 00_H 06_H 00_H 00_H 00_H 00_H 01_H 23_H, sendo:

- **00_H 03_H** - Canal do transdutor de destino.
- **03_H** - Classe do comando (Estado de operação do transdutor).
- **02_H** - Função do comando (Escrita do transdutor).
- **00_H 06_H** - Número de octetos.
- **00_H 00_H 00_H 00_H** - **Off-set**.
- **01_H** - Direção do motor de passo (00_H - horário, 01_H - anti-horário).
- **25_H** - Posição do motor de passo

Como resposta ao comando de controle do motor de passo, é enviado do STIM_1 para o NCAP o comando: 01_H 00_H 06_H 00_H 00_H 00_H 00_H 00_H 00_H

- **01_H** - Campo diferente de 0 - sucesso.
- **00_H 06_H** - Número de octetos no campo de dados.
- **00_H 00_H 00_H 00_H 00_H 00_H** - **Off-set**.

Para o controle do motor de passo 2, o NCAP envia para o STIM_1 o comando: 00_H 04_H 03_H 02_H 00_H 06_H 00_H 00_H 00_H 00_H 00_H 37_H, sendo:

- **00_H 03_H** - Canal do transdutor de destino.
- **03_H** - Classe do comando (Estado de operação do transdutor).
- **02_H** - Função do comando (Escrita do transdutor).
- **00_H 06_H** - Número de octetos.

- **00_H 00_H 00_H 00_H** - Off-set.

- **00_H** - Direção do motor de passo (00_H - horário, 01_H - anti-horário).

- **25_H** - Posição do motor de passo

Como resposta ao comando de controle do motor de passo 2, é enviado do STIM_1 para o NCAP o comando: 01_H 00_H 06_H 00_H 00_H 00_H 00_H 00_H

- **01_H** - Campo diferente de 0 - sucesso.

- **00_H 06_H** - Número de octetos no campo de dados.

- **00_H 00_H 00_H 00_H 00_H 00_H** - Off-set.

B.2 Comando para Monitoramento e Controle do Módulo STIM_2

O STIM_2 foram utilizados 3 transdutores com respectivos canais:

- **Canal 2** - Ventilador (**Cooler**)

- **Canal 3** - Ventilador (**Cooler**)

- **Canal 4** - Ventilador (**Cooler**)

Para o controle do ventilador 1, o NCAP envia para o STIM_1 o comando: 00_H 02_H 03_H 02_H 00_H 06_H 00_H 00_H 00_H 00_H 00_H, sendo:

- **00_H 02_H** - Canal do transdutor de destino.

- **03_H** - Classe do comando (Estado de operação do transdutor).

- **02_H** - Função do comando (Escrita do transdutor).

- **00_H 06_H** - Número de octetos.

- **00_H 00_H 00_H 00_H 00_H** - Off-set.

- **00_H** - Estado do atuador (00_H - Ligado, 01_H - Desligado).

Como resposta ao comando de controle do ventilador 1, é enviado do STIM_2 para o NCAP o comando: 01_H 00_H 06_H 00_H 00_H 00_H 00_H 00_H

- **01_H** - Campo diferente de 0 - sucesso.
- **00_H 06_H** - Número de octetos no campo de dados.
- **00_H 00_H 00_H 00_H 00_H 00_H** - **Off-set**.

Para o controle do ventilador 2, o NCAP envia para o STIM_1 o comando: **00_H 03_H 03_H 02_H 00_H 06_H 00_H 00_H 00_H 00_H 00_H 00_H**, sendo:

- **00_H 03_H** - Canal do transdutor de destino.
- **03_H** - Classe do comando (Estado de operação do transdutor).
- **02_H** - Função do comando (Escrita do transdutor).
- **00_H 06_H** - Número de octetos.
- **00_H 00_H 00_H 00_H 00_H** - **Off-set**.
- **00_H** - Estado do atuador (00_H - Ligado, 01_H - Desligado).

Como resposta ao comando de controle do ventilador 2, é enviado do STIM_2 para o NCAP o comando: **01_H 00_H 06_H 00_H 00_H 00_H 00_H 00_H 00_H**

- **01_H** - Campo diferente de 0 - sucesso.
- **00_H 06_H** - Número de octetos no campo de dados.
- **00_H 00_H 00_H 00_H 00_H 00_H** - **Off-set**.

Para o controle do ventilador 3, o NCAP envia para o STIM_1 o comando: **00_H 04_H 03_H 02_H 00_H 06_H 00_H 00_H 00_H 00_H 00_H 01_H**, sendo:

- **00_H 04_H** - Canal do transdutor de destino.
- **03_H** - Classe do comando (Estado de operação do transdutor).
- **02_H** - Função do comando (Escrita do transdutor).
- **00_H 06_H** - Número de octetos.
- **00_H 00_H 00_H 00_H 00_H** - **Off-set**.
- **01_H** - Estado do atuador (00_H - Ligado, 01_H - Desligado).

Como resposta ao comando de controle do ventilador 3, é enviado do STIM_2 para o NCAP o comando: $01_H 00_H 06_H 00_H 00_H 00_H 00_H 00_H$

- 01_H - Campo diferente de 0 - sucesso.
- $00_H 06_H$ - Número de octetos no campo de dados.
- $00_H 00_H 00_H 00_H 00_H 00_H$ - Off-set.

B.3 Comando para Monitoramento e Controle do Módulo WTIM_1

O WTIM_1 foram utilizados 2 transdutores com respectivos canais:

- **Canal 2** - sensor de temperatura LM35.
- **Canal 3** - Motor DC.

Para a leitura do sensor de temperatura, o NCAP envia para o WTIM_1 o comando: $00_H 02_H 03_H 01_H 00_H 06_H 00_H 00_H 00_H 00_H 00_H$, sendo:

- $00_H 02_H$ - Canal do transdutor de destino.
- 03_H - Classe do comando (Estado de operação do transdutor).
- 01_H - Função do comando (Leitura do transdutor).
- $00_H 06_H$ - Número de octetos.
- $00_H 00_H 00_H 00_H 00_H 00_H$ - Off-set.

Como resposta ao comando de requisição da temperatura, é enviado do WTIM_1 para o NCAP o comando: $01_H 00_H 06_H 00_H 00_H 00_H 00_H 66_H$, sendo:

- 01_H - Campo diferente de 0 - sucesso.
- $00_H 06_H$ - Número de octetos no campo de dados.
- $00_H 00_H 00_H 00_H 00_H$ - Off-set.
- 66_H - Valor da temperatura.

Para o controle do motor CC, o NCAP envia para o WTIM_1 o comando: $00_H 03_H 03_H 02_H 00_H 06_H 00_H 00_H 00_H 00_H 00_H$, sendo:

- $00_H 03_H$ - Canal do transdutor de destino.
- 03_H - Classe do comando (Estado de operação do transdutor).
- 02_H - Função do comando (Escrita do transdutor).
- $00_H 06_H$ - Número de octetos.
- $00_H 00_H 00_H 00_H 00_H$ - Off-set.
- 00_H - Estado do atuador (00_H - Ligado, 01_H - Desligado).

Como resposta ao comando de controle do motor CC, é enviado do WTIM_1 para o NCAP o comando: $01_H 00_H 06_H 00_H 00_H 00_H 00_H 00_H$

- 01_H - Campo diferente de 0 - sucesso.
- $00_H 06_H$ - Número de octetos no campo de dados.
- $00_H 00_H 00_H 00_H 00_H 00_H$ - Off-set.

B.4 Comando para Monitoramento e Controle do Módulo WTIM_2

O WTIM_2 foi utilizado 1 transdutor com o canal:

- **Canal 2** - sensor de temperatura LM35.

Para a leitura do sensor de temperatura, o NCAP envia para o WTIM_2 o comando: $00_H 02_H 03_H 01_H 00_H 06_H 00_H 00_H 00_H 00_H 00_H$, sendo:

- $00_H 02_H$ - Canal do transdutor de destino.
- 03_H - Classe do comando (Estado de operação do transdutor).
- 01_H - Função do comando (Leitura do transdutor).
- $00_H 06_H$ - Número de octetos.

- **00_H 00_H 00_H 00_H 00_H 00_H** - Off-set.

Como resposta ao comando de requisição da temperatura é enviado do WTIM_2 para o NCAP o comando: 01_H 00_H 06_H 00_H 00_H 00_H 00_H 00_H 54_H, sendo:

- **01_H** - Campo diferente de 0 - sucesso.
- **00_H 06_H** - Número de octetos no campo de dados.
- **00_H 00_H 00_H 00_H 00_H** - Off-set.
- **54_H** - Valor da temperatura.

B.5 Comandos de Requisição e Resposta dos TEDS

Os 4 TIMs desenvolvidos neste trabalho utilizam o mesmos comandos para requisição dos TEDS. As tabelas TEDS descritas neste trabalho foram 4, sendo: Meta-TEDS, TransducerChannel TEDS, User´s TEDS e PHY TEDS. Para requisição da tabela Meta-TEDS, o NCAP envia para o TIM o comando: 00_H 00_H 01_H 02_H 00_H 06_H 01_H 00_H 00_H 00_H 00_H 00_H, sendo:

- **00_H 00_H** - TIM de destino.
- **01_H** - Classe do comando (Comando comum para o TIM e o canal de transdutor).
- **02_H** - Função do comando (Leitura dos TEDS).
- **00_H 06_H** - Número de octetos.
- **01_H** - Indica a tabela que será feita a leitura (01_H - Meta-TEDS).
- **00_H 00_H 00_H 00_H 00_H** - Off-set.

A resposta do TIM para o NCAP é realizada através do comando: 01_H XX_H XX_H <Meta-TEDS>, sendo:

- **01_H** - Campo diferente de 0 - sucesso.
- **XX_H XX_H** - Número de octetos no campo de dados.
- **Meta-TEDS** - Campo com o conteúdo da tabela Meta-TEDS armazenada no TIM.

Para requisição da tabela TransducerChannel TEDS, o NCAP envia para o TIM o comando: $00_H 00_H 01_H 02_H 00_H 06_H 03_H 00_H 00_H 00_H 00_H$, sendo:

- $00_H 00_H$ - TIM de destino.
- 01_H - Classe do comando (Comando comum para o TIM e o canal de transdutor).
- 02_H - Função do comando (Leitura dos TEDS).
- $00_H 06_H$ - Número de octetos.
- 03_H - Indica a tabela que será feita a leitura (03_H - TransducerChannel TEDS).
- $00_H 00_H 00_H 00_H 00_H$ - Off-set.

A resposta do TIM para o NCAP é realizada através do comando: $01_H XX_H XX_H$ <TransducerChannel TEDS>, sendo:

- 01_H - Campo diferente de 0 - sucesso.
- $XX_H XX_H$ - Número de octetos no campo de dados.
- **TransducerChannel** - Campo com o conteúdo da tabela TransducerChannel TEDS armazenada no TIM.

Para requisição da tabela User's TEDS, o NCAP envia para o TIM o comando: $00_H 00_H 01_H 02_H 00_H 06_H 0C_H 00_H 00_H 00_H 00_H$, sendo:

- $00_H 00_H$ - TIM de destino.
- 01_H - Classe do comando (Comando comum para o TIM e o canal de transdutor).
- 02_H - Função do comando (Leitura dos TEDS).
- $00_H 06_H$ - Número de octetos.
- $0C_H$ - Indica a tabela que será feita a leitura ($0C_H$ - User's TEDS).
- $00_H 00_H 00_H 00_H 00_H$ - Off-set.

A resposta do TIM para o NCAP é realizada através do comando: $01_H XX_H XX_H$ <User's TEDS>, sendo:

- **01_H** - Campo diferente de 0 - sucesso.
- **XX_H XX_H** - Número de octetos no campo de dados.
- **User's TEDS** - Campo com o conteúdo da tabela User's TEDS armazenada no TIM.

Para requisição da tabela PHY TEDS, o NCAP envia para o TIM o comando: 00_H 00_H 01_H 02_H 00_H 06_H 0D_H 00_H 00_H 00_H 00_H, sendo:

- **00_H 00_H** - TIM de destino.
- **01_H** - Classe do comando (Comando comum para o TIM e o canal de transdutor).
- **02_H** - Função do comando (Leitura dos TEDS).
- **00_H 06_H** - Número de octetos.
- **0D_H** - Indica a tabela que será feita a leitura (0D_H - PHY TEDS).
- **00_H 00_H 00_H 00_H 00_H** - Off-set.

A resposta do TIM para o NCAP é realizada através do comando: 01_H XX_H XX_H <PHY TEDS>, sendo:

- **01_H** - Campo diferente de 0 - sucesso.
- **XX_H XX_H** - Número de octetos no campo de dados.
- **PHY TEDS** - Campo com o conteúdo da tabela PHY TEDS armazenada no TIM.

Transducer Electronic Data Sheet

Neste apêndice apresentam-se os TEDS descritas para o reconhecimento dos módulos TIMs pelo NCAP, de acordo com cada módulo desenvolvido (STIM_1, STIM_2, WTIM_1, WTIM_2) e o que cada campo representa dentro do contexto do padrão IEEE 1451.0-2007.

C.1 STIM_1

C.1.1 Meta-TEDS

A Meta-TEDS tem como objetivo disponibilizar todas as informações necessárias para obter acesso a qualquer canal de transdutor e definir uma identificação única para o TIM. Na Tabela C.1 apresentam-se os campos obrigatórios das Meta-TEDS que foram preenchidos como: (IEEE, 2007c)

- **Lenght** - número de octetos utilizados para descrever as Meta-TEDS incluindo, Campo, Comprimento (Comp.) e os 2 (dois) octetos do **Checksum**.
- **TEDSID** - identificação dos TEDS, representado por 4 octetos. O primeiro octeto indica que esse TEDS faz parte da norma IEEE 1451.0. O segundo octeto representa os TEDS que estão sendo descritos e que estão sendo representados nesta tabela, as Meta-TEDS. O terceiro octeto trata da identificação da versão TEDS, sendo que, para estes TEDS, trabalha com a versão atual, definida pela norma como valor 1. O último octeto representa o número de octetos do campo comprimento.
- **UUID (Universal Unique Identification)** - é um campo de identificação associado ao TIM, no qual o valor é único. O campo é representado por 10 octetos longos, e consiste em quatro subcampos: localização, fabricante, ano e tempo. Neste

contexto, para representação de acordo com o padrão IEEE 1451.0, utilizam-se 42 bits iniciais para representar a localização, sendo que, o MSB fixado em 1₂ representa o Norte e 0₂ o Sul. Os próximos 20 bits mais significativos representam a latitude em arcossegundos, utilizando números inteiros. O próximo bit fixado em 1₂ representa o Leste e 0₂ o Oeste. Os 20 bits restantes representam a longitude, utilizando números inteiros para representar o arcossegundos. O campo fabricante é representado por 4 bits, por haver apenas um fabricante, este campo é 0. O ano é representado pelos próximos 12 bits. Os 22 bits restantes representam a identificação do TIM, um código único de identificação desenvolvida pelo fabricante. Há uma determinada liberdade para o fabricante, então, o fabricante utiliza o seguinte campo de tempo exemplificado pela norma: dia do ano x 1000+sequência de produção.

- **OHoldOff** - um comando enviado do NCAP para o TIM sem resposta deve ser representado como uma operação falha. Valor que é representado em ponto flutuante, de acordo com a norma IEEE 754. Neste trabalho, define-se 5 segundos, após, o comando é considerado falho.
- **TestTime** - tempo para a realização do auto teste do módulo TIM.
- **MaxChan** - quantidade de canais de transdutores implementados no TIM.
- **Checksum** - representa a soma de todos os octetos dentro da Meta-TEDS, sendo representado por 2 octetos.

Tabela C.1 – Meta-TEDS.

Campo (Hex.)	Nome	Tipo	Tamanho (Int.)	Valor (Hex.)
—	Lenght	UInt32	4	0x00 0x00 0x00 0x25
0x03	TEDSID	UInt	4	0x00 0x01 0x01 0x01
0x04	UUID	UUID	10	0x08 0xFB 0x61 0xB4 0x80 0x81 0xF6 0x43 0xA1 0xB1
0x0A	OHoldOff	Float32	4	0x40 0xA0 0x00 0x00
0x0C	TestTime	Float32	4	0x40 0x00 0x00 0x00
0x0D	MaxChan	UInt16	2	0x00 0x03
—	Checksum	UInt16	2	0x07 0x35

Fonte: Santos Filho (2012)

C.1.2 TransducerChanel TEDS

O TransducerChanel TEDS descrito nos TEDS será para disponibilizar todas as informações de operação e endereçamento do canal de transdutor. O TransducerChanel

TEDS é acessado usando comandos de requisição, comandos de leitura dos TEDS, escrita dos TEDS ou atualização. Os TEDS podem ser implementadas apenas para leitura prevenindo que mudanças sejam feitas, pois, poderiam causar incompatibilidade, neste caso, os comandos escrita e atualização não são aplicados. Nas Seções C.1.2.1, C.1.2.3 e C.1.2.2 apresentam-se os campos utilizados para descrição do sensor de temperatura LM35, do motor de passo 1 e motor de passo 2 respectivamente.

C.1.2.1 Sensor LM35

Na Tabela C.2 apresenta-se a estrutura do TransducerChanel TEDS referente ao sensor de temperatura LM35 canal 1 e o nome dos campos que foram definidos são(IEEE, 2007c):

- **Lenght** - número de octetos utilizados para descrever as TransducerChanel TEDS, incluindo Campo, Comprimento (Comp.), Valor e os 2 (dois) octetos do **Checksum**.
- **TEDSID** - identificação dos TEDS, campo semelhante à identificação das Meta-TEDS, porém, o segundo octeto é representado através do valor 3, que identifica TransducerChanel TEDS.
- **CalKey** - este módulo de transdutor não irá realizar nenhum tipo de calibração. Segundo o padrão, o valor para representar o comportamento é 0 (zero).
- **ChanType** - define a característica do transdutor: sensor (representado por: 0x00), atuador (representado por: 0x01) ou sensor de evento (representado por: 0x02).
- **PhyUnits** - utilizado para definir o tipo de unidade física descrita pelo canal. Este campo é subdividido em 10 subcampos, no qual, para sua representação, utiliza-se o Sistema Internacional (SI) de unidades. Quando o expoente dessas unidades for igual a 0_{10} , o campo deve receber o valor 128_{10} . Caso a unidade for fixada, deve-se utilizar o valor 130_{10} . Para a representação, foi utilizado um sensor de temperatura no qual se utiliza a unidade em Kelvin, sendo fixado o valor 130_{10} no campo 57_{10} . De acordo com IEEE (2007c) na Tabela 48, descrevem os tipos de unidades disponíveis. Para a representação em formato TLV, os campos seguem o seguinte padrão:
 - **Campo - 12:** representa a PhyUnits no TransducerChanel TEDS.
 - **Tamanho - 6:** quantidade de octetos para descrever os comandos na linha.
 - **Tipo de unidade - 50:** representa as unidades disponíveis no padrão.

- Tamanho - 1:** quantidade de octetos necessários para descrever o tipo de unidade física.
 - Estrutura da unidade física - 0:** representa a estrutura da unidade física utilizada para descrever o valor da unidade. Neste trabalho, utilizou-se a PULSL_UNITS, que representa o Sistema Internacional. Em (IEEE, 2007c), na Tabela 3, apresentam-se as estruturas das unidades físicas para a descrição da unidade.
 - Unidade física - 57:** tipo da unidade física utilizada. Neste trabalho, utilizou-se Kelvins.
 - Tamanho - 1:** número de octetos utilizados para representar de acordo com a estrutura da unidade física.
 - Valor - 128:** utilizado para definir o expoente da unidade física.
- LowLimit-** define o limite mínimo do sensor de temperatura. Para o sensor de temperatura descrito neste trabalho, foi utilizado 273,15 K.
 - HighLimit-** define o limite máximo do sensor de temperatura. Para o sensor de temperatura descrito neste trabalho, foi utilizado 328,15 K.
 - Oerror-** variação de erro do sensor de temperatura.
 - SelfTest-** para realizar o autoteste no sistema, fixa-se o valor 1. Caso o módulo TIM não possua a capacidade de autoteste, fixa-se o valor 0.
 - Sample-** este campo é representado por 3 (três) subcampos obrigatórios, cada um contendo 3 octetos. Os 3 (três) subcampos possuem o tipo, comprimento e o valor. O subcampo **Data Model** representado pelos 3 primeiros octetos, define o tipo de dados. Neste trabalho, definiu-se um inteiro de 10 bits. O subcampo **Data model lenght** representado pelos 3 octetos seguintes, define o tamanho do campo para realizar a representação específica. Os 3 últimos octetos representam o subcampo **More Significant Bits** que, neste contexto, utilizaram-se 8 bits, indicando a saída do conversor A/D (Analógico/Digital).
 - UpdateT-** este transdutor está no modo “free-running” e está atualizando as amostras a uma taxa de 10/s. Portanto, o valor deste campo é 0,1, que é o tempo máximo entre a recepção da leitura e a disponibilidade de uma nova disponível.
 - RSetupT-** este campo contém o tempo máximo (t_{rs}), expresso em segundos, entre o recebimento do **trigger** pelo módulo de transdutor e do tempo que os dados estão

disponíveis para ser lido. O tempo de configuração de leitura para este transdutor é de $0,25\mu\text{s}$.

Tabela C.2 – TransducerChannel TEDS - LM35.

Campo (Hex.)	Nome	Tipo	Tamanho (Int.)	Valor (Hex.)
—	Lenght	UInt32	4	0x00 0x00 0x00 0x60
0x03	TEDSID	UInt8	4	0x00 0x03 0x01 0x01
0x0A	CalKey	UInt8	1	0x00
0x0B	ChanType	UInt8	1	0x00
0x0C	PhyUnits	UNITS	6	0x32 0x01 0x00 0x39 0x01 0x82
0x0D	LowLimit	Float32	4	0x43 0x88 0x93 0x33
0x0E	HiLimit	Float32	4	0x43 0xA4 0x13 0x33
0x0F	Oerror	Float32	4	0x3F 0x00 0x00 0x00
0x10	SelfTest	UInt8	1	0x01
0x12	Sample	—	9	0x28 0x01 0x00 0x29 0x01 0x01 0x30 0x01 0x08
0x14	UpdateT	Float32	4	0x3D 0xCC 0xCC 0xCD
0x16	RSetupT	Float32	4	0x37 0xD1 0xB7 0x17
0x17	Speriod	Float32	4	0x3D 0xCC 0xCC 0xCD
0x18	WarmUpT	Float32	4	0x41 0xF0 0x00 0x00
0x19	Rdelay	Float32	4	0x37 0xD1 0xB7 0x17
0x20	TestTime	Float32	04	0x41 0xF0 0x00 0x00
0x1F	Sampling	—	3	0x31 0x01 0x02
—	Checksum	UInt16	2	0x11 0xBA

Fonte: Santos Filho (2012)

- **Speriod**- o período de amostragem mínimo do canal de transdutor é de 1 décimo de segundo desconsiderando o transdutor. Assim, foi atribuído o valor correspondente em ponto flutuante.
- **WarmUpT**- apresenta o tempo necessário em segundos para o TransducerChannel entrar em modo de operação.
- **RDelay**- pior caso de atraso dos dados existentes é 25ms.
- **TestTime**- o campo é necessário para todos TransducerChannel que implementa autoteste. Se o campo for omitido e implementar o autoteste, o NCAP considerará como erro.
- **Sampling**- o transdutor está sempre rodando em modo “**Free Running**” sem “**Pre-trigger**”. Nesta descrição dos TEDS, foi fixado o valor 2_{10} , de acordo com (IEEE, 2007c) descrito na Tabela 54.

- Checksum**- representa a soma de todos os octetos dentro do TransducerChanel TEDS no qual o valor é representado por 2 octetos.

C.1.2.2 Motor de passo 1

Para a representação do canal 2 referente ao motor de passo 1, foram definidos os campos:

- Lenght** - número de octetos utilizados para descrever as TransducerChanel TEDS incluindo, Campo, Comprimento (Comp.), Valor e os 2 (dois) octetos do **Checksum**.
- TEDSID** - identificação dos TEDS, campo semelhante à identificação das Meta-TEDS, porém, o segundo octeto é representado através do valor 3, que identifica TransducerChanel TEDS.
- CalKey** - este módulo de transdutor não irá realizar nenhum tipo de calibração. Segundo (IEEE, 2007c), o valor para representar o comportamento é 0 (zero).
- ChanType** - define a característica do transdutor: sensor (representado por: 0x00), atuador (representado por: 0x01) ou sensor de evento (representado por: 0x02).
- PhyUnits** - utilizado para definir o tipo de unidade física descrita pelo canal. De acordo com (IEEE, 2007c) na Tabela 48 descrevem os tipos de unidades disponíveis. Para a representação da unidade física do motor de passo foram utilizados:
 - Campo - 12:** representa a PhyUnits no TransducerChanel TEDS.
 - Tamanho - 6:** quantidade de octetos para descrever os comandos na linha (tupla).
 - Tipo de unidade - 50:** representa as unidades disponíveis no padrão, outras unidades podem ser visualizadas em (IEEE, 2007c) na Tabela 2.
 - Tamanho - 1:** quantidade de octetos necessários para descrever o tipo de unidade física.
 - Estrutura da unidade física - 0:** representa a estrutura da unidade física utilizada para descrever o valor da unidade. Neste trabalho, utilizou-se a PULSLUNITS, que representa o Sistema Internacional. Em (IEEE, 2007c) na Tabela 3 apresentam-se as estruturas das unidades físicas para a descrição da unidade.

- Unidade física - 57:** tipo da unidade física utilizada. Neste trabalho, utilizaram-se radianos para especificar o deslocamento do motor de passo.
- Tamanho - 1:** número de octetos utilizados para representação de acordo com a estrutura da unidade física.
- Valor - 128:** utilizado para definir o expoente da unidade física.
- LowLimit** - representa o valor mínimo da unidade que será medida ou controlada. Para o motor de passo, foi utilizado 0.
- HighLimit** - representa o valor máximo da unidade que será medida ou controlada. Para o motor de passo, foi definido 360.
- Oerror** - variação de erro da unidade física a ser medida ou controlada. Para o motor de passo, foi definido 1.8.
- SelfTest** - campo que determina o tempo limite para realizar o auto-teste no sistema. Para realizar o autoteste fixa-se 1, para não realizar o autoteste, fixa-se 0.
- Sample** - este campo é representado por 3 (três) subcampos obrigatórios, cada um contendo 3 octetos. Os 3 (três) subcampos possuem o tipo, comprimento e o valor. Para o motor de passo, foram definidos 4 bits para a comunicação.
- UpdateT** - este campo contém o tempo máximo expresso em segundos entre o disparo de um evento e a conclusão da primeira amostra dos dados fixos. Se o intervalo de amostragem é programável e o TransducerChannel está sendo operado em um dos modos de amostragem “**free-running**” este valor não é relevante e deve ser definido como zero.
- WSetup** - este campo é obrigatório para atuadores, porém, para sensores, pode ser omitido. O campo WSetup contém o tempo mínimo expresso em segundos entre o fim da escrita de um **frame** e o disparo da aplicação.
- SPeriod** - apresenta o período de amostragem mínima expressa em segundos do canal de transdutor livre da leitura/escrita, estando limitado no tempo de conversão dos A/D ou D/A, velocidade do processador e alguns sistemas mais complexos.
- WarmUpt** - apresenta o tempo necessário em segundos para o TransducerChannel entrar em modo de operação.
- TestTime** - tempo necessário para realizar o **self-test**. Caso o campo **self-test** foi fixado com 0, o campo pode ser omitido.

- **Sampling** - campo obrigatório para todas os TEDS. O campo **sampling** é utilizado para descrever qual tipo de amostragem é suportado pelo TransducerChannel. Nesta descrição dos TEDS, foi fixado o valor 0_{10} de acordo com (IEEE, 2007c), descrito na Tabela 54.
- **EndOfSet** - descreve o modo de operação que o atuador pode suportar. Neste contexto, foi definido o valor 1_{10} que especifica que o TransducerChannel detém o último valor até que outro dado seja recebido. Em (IEEE, 2007c), Tabela 57, descrevem outras opções para definir o modo de operação do TransducerChannel.
- **DataXmit** - descreve o modo de transmissão dos dados em que o canal de transdutor suporta. Nesta descrição, o TransducerChannel foi definido para operar somente em modo de comando. Em (IEEE, 2007c), Tabela 58, descrevem outras opções para definir o modo transmissão do TransducerChannel.
- **ActHalt** - define o modo de parada do atuador. Para este campo, foi fixado 2_{10} , no qual define a parada após finalizar a aplicação corrente, mantendo o valor de saída até retornar ao estado de operação ou entrar em estado de dormência.
- **Checksum** - representa a soma de todos os octetos dentro do TransducerChanel TEDS no qual o valor é representado por 2 octetos.

Na Tabela C.3, apresenta-se a tabela dos TEDS referente ao motor de passo 1 canal 2.

C.1.2.3 Motor de passo 2

Para a representação do canal 3, referente ao motor de passo 2, foram definidos os campos de acordo com a Seção C.1.2.2. A descrição dos TEDS referente ao motor de passo 2 canal 3 são as mesmas apresentada na Tabela C.3.

C.1.3 User's Transducer Name TEDS

O User's Transducer Name TEDS descrito nos TEDS é necessário para todos os TIMs. O User's Transducer Name TEDS disponibiliza um espaço para o armazenamento do nome no qual o sistema ou o usuário final saberá qual será o transdutor. O User's Transducer Name TEDS pode ser associado com o TIM ou com o TransducerChanel TEDS que são acessados usando requisições através de comando escrita, leitura ou uma atualização. Para

Tabela C.3 – TransducerChannel TEDS - Motor de passo 1.

Campo (Hex.)	Nome	Tipo	Tamanho (Int.)	Valor (Hex.)
—	Lenght	UInt32	4	0x00 0x00 0x00 0x63
0x03	TEDSID	UInt8	4	0x00 0x03 0x01 0x01
0x0A	CalKey	UInt8	1	0x00
0x0B	ChanType	UInt8	1	0x01
0x0C	PhyUnits	UNITS	6	0x32 0x01 0x00 0x33 0x01 0x82
0x0D	LowLimit	Float32	4	0x00 0x00 0x00 0x00
0x0E	HiLimit	Float32	4	0x43 0xB4 0x00 0x00
0x0F	Oerror	Float32	4	0x3F 0xE6 0x66 0x66
0x10	SelfTest	UInt8	1	0x01
0x12	Sample	—	9	0x28 0x01 0x00 0x29 0x01 0x01 0x30 0x01 0x08
0x14	UpdateT	Float32	4	0x3D 0xCC 0xCC 0xCD
0x16	RSetupT	Float32	4	0x37 0xD1 0xB7 0x17
0x17	Speriod	Float32	4	0x3D 0xCC 0xCC 0xCD
0x18	WarmUpT	Float32	4	0x41 0xF0 0x00 0x00
0x19	TestTime	Float32	4	0x41 0xF0 0x00 0x00
0x1F	Sampling	—	3	0x31 0x01 0x00
0x21	EnOfSet	UInt8	1	0x01
0x22	DataXmit	UInt8	1	0x01
0x24	ActHalt	UInt8	1	0x01
—	Checksum	UInt16	2	0x10 0x20

Fonte: Santos Filho (2012)

descrever a tabela User's Transducer Name TEDS, foi utilizado o nome "STIM.1". Os campos do User's Transducer Name TEDS são definidos como (IEEE, 2007c):

- **Lenght** - número de octetos utilizados para descrever as User's Transducer Name TEDS incluindo, Campo, Comprimento (Tamanho), Valor e os 2 (dois) octetos do Checksum.
- **TEDSID** - identificação dos TEDS, campo semelhante à identificação das Meta-TEDS e TransducerChanel, porém, o segundo octeto é representado através do valor 12_{10} que identifica o User's Transducer Name TEDS.
- **Format** - define o formato da representação do campo TCName, sendo que, 0_{10} é uma representação definida pelo usuário de acordo com (IEEE, 2007c) na Tabela 74.
- **TCName** - descreve o nome do módulo TIM e o canal de transdutor de acordo com o padrão definido no campo Format. Neste trabalho, utilizou-se o padrão ASCII

(**American Standard Code for Information Interchange**), para definir o nome do módulo TIM como STIM_1.

- **Checksum** - representa a soma de todos os octetos dentro do User's Transducer Name TEDS, no qual o valor é representado por 2 octetos.

Na Tabela C.4, especificam-se os campos User's Transducer Name TEDS.

Tabela C.4 – User's Transducer Name TEDS.

Campo (Hex.)	Nome	Tipo	Tamanho (Int.)	Valor (Hex.)
-	Lenght	UInt32	4	0x00 0x00 0x00 0x19
0x03	TEDSID	UInt32	4	0x00 0x0C 0x01 0x01
0x04	Format	UInt8	1	0x00
0x05	TCName	UInt8	11	0x53 0x54 0x49 0x4D 0x31 0x5F 0x52 0x53 0x32 0x33 0x32
—	Checksum	UInt16	2	0x03 0x4F

Fonte: Santos Filho (2012)

C.1.4 PHY TEDS

As PHY TEDS são necessárias em todos os TIMs e são baseadas de acordo com cada norma de interfaceamento definida pelo padrão IEEE 1451. A característica básica das PHY TEDS é disponibilizar as informações necessárias da interface de comunicação. Os octetos da PHY TEDS são constantes e apenas para leitura. É importante destacar que a norma IEEE 1451.0-2007 não é descrita para o padrão IEEE 1451.4. Para maiores informações, a norma IEEE 1451.4 (IEEE, 2004) deve ser consultada. Neste trabalho, a PHY TEDS foi descrita baseada no padrão IEEE 1451.2 e foi utilizado o padrão de interfaceamento RS-232, com as seguintes configurações: 4800 bps, 8 bits, N Paridade, 1 **Stop** bit. O padrão IEEE P1451.2, com base na interface RS-232, encontra-se em fase de atualização, no entanto, trabalhos com a nova especificação das descrição das PHY TEDS estão sendo desenvolvidos, como, por exemplo em (IEEE, 2007c) e (SONG; LEE, 2008b), autores que fazem parte do comitê IEEE 1451. Na Tabela C.5, apresenta-se a descrição utilizada neste trabalho para a tabela PHY TEDS baseados nos trabalhos (IEEE, 2007c) e (SONG; LEE, 2008b).

Tabela C.5 – PHY TEDS.

Campo (Hex.)	Nome	Tipo	Tamanho (Int.)	Valor (Hex.)
-	Lenght	UInt32	4	0x00 0x00 0x00 0x55
0x03	TEDSID	UInt32	4	0x00 0x0C 0x01 0x1
0x0A	RS232	UInt8	1	0x01
0x0C	MaxBPS	UInt32	4	0x00 0x00 0x04 0xB0
0x0D	MaxCDev	UInt16	2	0x00 0x01
0x0E	Encrypt	UInt16	2	0x00 0x00
0x0F	Authent	Boolean	1	0x00
0x10	MinKeyL	UInt16	2	0x00 0x00
0x11	MaxkeyL	UInt16	2	0x00 0x00
0x12	MaxSDU	UInt16	2	0x00 0x01
0x13	MinALat	UInt32	4	0x00 0x00 0x00 0x05
0x14	MinTLat	UInt32	4	0x00 0x00 0x00 0x05
0x15	MaxXact	UInt8	1	0x01
0x16	Battery	UInt8	1	0x01
0x17	Version	UInt16	2	0x00 0x00
0x18	MaxRetry	UInt16	2	0x00 0x05
0x29	Baud	UInt32	4	0x00 0x00 0x12 0xC0
0x2A	DataBits	UInt8	1	0x08
0x2B	Parity	UInt8	1	0x00
0x2C	StopBit	UInt8	1	0x01
0x2D	Terminator	UInt8	1	0x00
	Checksum	UInt16	2	0x04 0xB0

Fonte: Santos Filho (2012)

C.2 STIM_2

Nesta seção apresenta-se a descrição dos TEDS referente aos 3 atuadores utilizado no STIM_2.

C.2.1 Meta-TEDS

Na Tabela C.6, apresenta-se os campos obrigatórios das Meta-TEDS, referentes ao STIM_2. Os campos utilizados são os mesmos aplicados no STIM_1 e descritos na Seção C.1.1.

C.2.2 TransducerChanel TEDS

Nesta seção apresenta-se a descrição dos TEDS referente aos 3 atuadores utilizados no STIM_2.

Tabela C.6 – Meta-TEDS.

Campo (Hex.)	Nome	Tipo	Tamanho (Int.)	Valor (Hex.)
—	Lenght	UInt32	4	0x00 0x00 0x00 0x25
3	TEDSID	UInt	4	0x00 0x01 0x01 0x01
4	UUID	UUID	10	0x08 0xFB 0x61 0xB4 0x80 0x81 0xF6 0x44 0xC2 0xC2
10	OHoldOff	Float32	4	0x40 0xA0 0x00 0x00
12	TestTime	Float32	4	0x40 0x40 0x00 0x00
13	MaxChan	UInt16	2	0x00 0x03
—	Checksum	UInt16	2	0x07 0xA4

Fonte: Santos Filho (2012)

C.2.2.1 Ventilador 1

Na Tabela C.7, apresentam-se os campos obrigatórios do TransducerChannel TEDS referente ventilador 1 utilizado como atuador no STIM_2. Os campos utilizados para a descrição do ventilador 1 são os mesmos valores apresentados para o atuador do STIM_1 e podem ser visualizados na Seção C.1.2.2.

C.2.2.2 Ventilador 2

Para a representação do canal 3, referente ao ventilador 2, foram definidos os campos de acordo com a Seção C.1.2.2. A descrição dos TEDS referente ao ventilador 2 canal 3 são os mesmos valores apresentados na Tabela C.7.

C.2.2.3 Ventilador 3

Para a representação do canal 4, referente ao ventilador 3, foram definidos os campos de acordo com a Seção C.1.2.2. A descrição dos TEDS referente ao ventilador 3 canal 4 são os mesmos valores apresentados na Tabela C.7.

C.2.3 User's Transducer Name TEDS

Os campos utilizados para descrição das User's Transducer Name TEDS são os mesmos apresentados na Seção C.1.3. Na Tabela C.8, especificam-se os campos User's Transducer Name TEDS.

Tabela C.7 – TransducerChannel TEDS - Ventilador 1.

Campo (Hex.)	Nome	Tipo	Tamanho (Int.)	Valor (Hex.)
—	Lenght	UInt32	4	0x00 0x00 0x00 0x63
0x03	TEDSID	UInt8	4	0x00 0x03 0x01 0x01
0x0A	CalKey	UInt8	1	0x00
0x0B	ChanType	UInt8	1	0x01
0x0C	PhyUnits	UNITS	6	0x32 0x01 0x00 0x38 0x01 0x82
0x0D	LowLimit	Float32	4	0x00 0x00 0x00 0x00
0x0E	HiLimit	Float32	4	0x41 0x40 0x00 0x00
0x0F	Oerror	Float32	4	0x3F 0x80 0x00 0x00
0x10	SelfTest	UInt8	1	0x01
0x012	Sample	—	9	0x28 0x01 0x00 0x29 0x01 0x01 0x30 0x01 0x08
0x14	UpdateT	Float32	4	0x3D 0xCC 0xCC 0xCD
0x16	RSetupT	Float32	4	0x3D 0xCC 0xCC 0xCD
0x17	Speriod	Float32	4	0x3D 0xCC 0xCC 0xCD
0x18	WarmUpT	Float32	4	0x41 0xF0 0x00 0x00
0x19	Rdelay	Float32	4	0x41 0xF0 0x00 0x00
0x1F	Sampling	—	3	0x31 0x01 0x00
0x21	EnOfSet	UInt8	1	0x01
0x22	DataXmit	UInt8	1	0x01
0x23	ActHalt	UInt8	1	0x01
—	Checksum	UInt16	2	0x0F 0x4A

Fonte: Santos Filho (2012)

C.2.4 PHY TEDS

As PHY TEDS descritas para o STIM_2 são os mesmos campos e os mesmos valores especificados para o STIM_1 apresentado na Seção C.1.4, Tabela C.5.

C.3 WTIM_1

Nesta seção apresenta-se a descrição dos TEDS referente 1 sensor utilizado no WTIM_1.

C.3.1 Meta-TEDS

Na Tabela C.9, apresentam-se os campos obrigatórios das Meta-TEDS referentes ao WTIM_1. Os campos utilizados são os mesmos aplicados no STIM_1 e descritos na Seção

Tabela C.8 – User’s Transducer Name TEDS.

Campo (Hex.)	Nome	Tipo	Tamanho (Int.)	Valor (Hex.)
-	Lenght	UInt32	4	0x00 0x00 0x00 0x19
0x03	TEDSID	UInt32	4	0x00 0x0C 0x01 0x1
0x04	Format	UInt8	1	0x00
0x05	TCName	UInt8	11	0x53 0x54 0x49 0x4D 0x32 0x5F 0x52 0x53 0x32 0x33 0x32
—	Checksum	UInt16	2	0x03 0x51

Fonte: Santos Filho (2012)

C.1.1.

Tabela C.9 – Meta-TEDS.

Campo (Hex.)	Nome	Tipo	Tamanho (Int.)	Valor (Hex.)
—	Lenght	UInt32	4	0x00 0x00 0x00 0x25
0x03	TEDSID	UInt	4	0x00 0x01 0x01 0x01
0x04	UUID	UUID	10	0x08 0xFB 0x61 0xB4 0x80 0x81 0xF6 0x41 0xA1 0xFB
0x0A	OHoldOff	Float32	4	0x40 0xA0 0x00 0x00
0x0B	TestTime	Float32	4	0x3E 0xE6 0x66 0x66
0x0C	MaxChan	UInt16	2	0x00 0x01
—	Checksum	UInt16	2	0x09 0x2B

Fonte: Santos Filho (2012)

C.3.2 TransducerChanel TEDS

Nesta seção, apresenta-se a descrição dos TEDS referente a 1 sensor e 1 atuador utilizado no WTIM.1.

C.3.2.1 Sensor LM35

Para a representação do canal 1, referente ao sensor LM35, foram definidos os campos de acordo com a Seção C.1.2. A descrição dos TEDS referente ao sensor LM35 são os mesmos valores apresentados na Tabela C.2.

C.3.3 User’s Transducer Name TEDS

Os campos utilizados para descrição das User’s Transducer Name TEDS são os mesmos apresentados na Seção C.1.3. Na Tabela C.10, especificam-se os campos User’s Transducer Name TEDS.

Tabela C.10 – User’s Transducer Name TEDS.

Campo (Hex.)	Nome	Tipo	Tamanho (Int.)	Valor (Hex.)
-	Lenght	UInt32	4	0x00 0x00 0x00 0x19
0x03	TEDSID	UInt32	4	0x00 0x0C 0x01 0x1
0x04	Format	UInt8	1	0x00
0x05	TCName	UInt8	12	0x57 0x54 0x49 0x4D 0x31 0x5F 0x5A 0x69 0x67 0x42 0x65 0x65
—	Checksum	UInt16	2	0x04 0x4F0

Fonte: Santos Filho (2012)

C.3.4 PHY TEDS

O WTIM faz parte do padrão IEEE 1451.5 e define uma interface de comunicação utilizando as redes sem fio. Neste trabalho, as PHY TEDS foram descritas baseadas no padrão ZigBee, em que a norma define os campos de 3 a 24 como padrões para qualquer interface sem fio e do campo 64 ao 80 especificado de acordo com o padrão de cada interface (IEEE, 2007b). Os campos padrões da PHY TEDS são:

- **Lenght** - número de octetos utilizados para descrever as PHY TEDS incluindo, Campo, Comprimento (Tamanho), Valor e os 2 (dois) octetos do Checksum.
- **TEDSID** - identificação dos TEDS.
- **Radio** - identificação da rede sem fio.
- **MaxBPS** - quantidade máxima de bits por segundo.
- **MaxCDev** - quantidade máxima de dispositivos conectados.
- **MaxRDev** - número máximo de dispositivos que poderia ser operacional simultaneamente com este dispositivo.
- **Encrypt** - tipo de criptografia e tamanho da chave.
- **Authent** - autenticação.
- **MinKeyL** - tamanho mínimo da chave para funções de segurança.
- **MaxBPS** - tamanho máximo da chave para funções de segurança.
- **MaxSDU** - tamanho máximo para transferência entre os dispositivos.
- **MinALat** - período mínimo para inicializar uma primeira transmissão para um dispositivo não conectado.

- **MTLat** - o período de transmissão de N-octetos, em que N é o tamanho mínimo.
- **MaxXact** - número máximo de comandos na fila não completados.
- **Battery** - um valor diferente de 0 (zero) indica que o circuito é alimentado por bateria.
- **RadioVer** - um valor 0 (Zero) indica versão não conhecida.
- **MaxRetry** - número máximo de tentativas antes de desconectar.

Na Tabela C.11 apresenta-se a descrição utilizada neste projeto para a tabela PHY TEDS.

Tabela C.11 – PHY TEDS.

Campo (Hex.)	Nome	Tipo	Tamanho (Int.)	Valor (Hex.)
-	Lenght	UInt32	4	0x00 0x00 0x00 0x4F
0x03	TEDSID	UInt32	4	0x05 0x0C 0x00 0x01
0x0A	Radio	UInt8	1	0x01
0x0B	MaxBPS	UInt32	4	0x00 0x00 0x00 0xFA
0x0C	MaxCDev	UInt16	2	0x00 0x01
0x0D	MaxRDev	UInt16	2	0x00 0x01
0x0E	Encrypt	UInt16	2	0x00 0x00
0x0F	Authent	Boolean	1	0x00
0x12	MaxSDU	UInt16	2	0x00 0xFF
0x13	MinALat	UInt32	4	0x00 0x00 0x04 0x00
0x14	MinTLat	UInt32	4	0x00 0x00 0x04 0x00
0x15	MaxXact	UInt8	1	0x01
0x16	Battery	UInt8	1	0x00
0x17	Version	UInt16	2	0x00 0x00
0x18	MaxRetry	UInt16	2	0x00 0x00
0x40	Phy_Ch	UInt32	2	0x00 0x0B
0x41	Phy_ch_w	UInt16	2	0x00 0x0F
0x43	phyFreq	UInt8	2	0x09 0xB0
0x44	RangeMax	UInt16	2	0x05 0x7C
—	Checksum	UInt16	2	0x69 0x0D

Fonte: Santos Filho (2012)

- **Phy_Ch** - número de canais que o ZigBee suporta.
- **Phy_ch_w** - canal usado.
- **PhyFrec** - frequência do ZigBee.
- **RangeMax** - distância entre os módulos de comunicação.
- **Checksum** - representa a soma de todos os octetos dentro da PHY TEDS.

C.4 WTIM_2

Nesta seção, apresenta-se a descrição dos TEDS referente aos 2 transdutores utilizados no WTIM_2.

C.4.1 Meta-TEDS

Na Tabela C.12, apresentam-se os campos obrigatórios das Meta-TEDS referentes ao WTIM_2. Os campos utilizados são os mesmos aplicados no STIM_1 e descritos na Seção C.1.1.

Tabela C.12 – Meta-TEDS.

Campo	Nome	Tipo	Tamanho	Valor
—	Lenght	UInt32	4	0x00 0x00 0x00 0x25
0x03	TEDSID	UInt	4	0x00 0x01 0x01 0x01
0x04	UUID	UUID	10	0x08 0xFB 0x61 0xB4 0x80 0x81 0xF6 0x43 0x67 0x1C
0x0A	OHoldOff	Float32	4	0x40 0xA0 0x00 0x00
0x0C	TestTime	Float32	4	0x3F 0x99 0x99 0x9A
0x0D	MaxChan	UInt16	2	0x00 0x02
—	Checksum	UInt16	2	0x08 0x30

Fonte: Santos Filho (2012)

C.4.2 TransducerChanel TEDS

Nesta seção, apresenta-se a descrição dos TEDS referente a 1 sensor utilizado no WTIM_2.

C.4.2.1 Sensor LM35

Para a representação do canal 1, referente ao sensor LM35, foram definidos os campos de acordo com a Seção C.1.2. A descrição dos TEDS referente ao sensor LM35 são os mesmos valores apresentados na Tabela C.2.

C.4.2.2 Motor CC

Para a representação do canal 2, referente ao motor CC, foram definidos os campos de acordo com a Seção C.1.2.2. A descrição dos TEDS referente ao motor CC canal 2 são os mesmos valores apresentados na Tabela C.7.

C.4.3 User's Transducer Name TEDS

Os campos utilizados para descrição das User's Name TEDS são os mesmos apresentados na Seção C.1.3. Na Tabela C.13, especificam-se os campos User's Transducer Name TEDS.

Tabela C.13 – User's Transducer Name TEDS.

Campo (Hex.)	Nome	Tipo	Tamanho (Int.)	Valor (Hex.)
-	Lenght	UInt32	4	0x00 0x00 0x00 0x19
0x03	TEDSID	UInt32	4	0x00 0x0C 0x01 0x1
0x04	Format	UInt8	1	0x00
0x05	TCName	UInt8	12	0x57 0x54 0x49 0x4D 0x32 0x5F 0x5A 0x69 0x67 0x42 0x65 0x65
—	Checksum	UInt16	2	0x04 0x50

Fonte: Santos Filho (2012)

C.4.4 PHY TEDS

Os campos utilizados para descrição das PHY TEDS são os mesmos apresentados na Seção C.3.4. Na Tabela C.11 especifica-se os campos PHY TEDS referente a interface ZigBee utilizada neste trabalho.

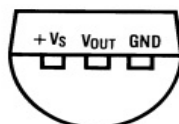
Apêndice D

Descrição dos Transdutores

D.1 LM35

O sensor LM35 é fabricado pela National Semiconductor que apresenta uma tensão linear relativa à temperatura em °C (graus Celsius), sendo que, para cada 10mV, representa 1°C. O sensor LM35 opera em uma faixa de 4V a 30V, variando a temperatura entre -55°C e 150°C, sem necessidade de calibração externa e com precisão de 0,25°C. Na Figura D.1, apresenta-se o sensor LM35 visualizado de cima.

Figura D.1 – Sensor LM35 visualizado de cima.



Fonte: Santos Filho (2012)

D.2 Motor de Passo

Os motores de passo são dispositivos mecânicos eletromagnéticos que podem ser controlados digitalmente, utilizando um **hardware** específico ou um **software** e são controlados por uma série de campos eletromagnéticos que são ativados e desativados eletronicamente. Dispositivos que são utilizados em aparelhos em que a precisão é um fator de grande importância, como, por exemplo: impressoras, **plotters**, **scanners**, **drivers** de disquetes, discos rígidos etc. As vantagens dos motores de passo são: seguem uma lógica digital, alta precisão em seu posicionamento, precisão no torque aplicado e resposta rápida de aceleração e desaceleração. As desvantagens são: baixo desempenho em altas velocidades e requerem certo grau de complexidade para ser operado.

Para o controle do motor de passo, as bobinas devem ser energizadas. A sequência para energizar as bobinas, para realizar o passo completo, é apresentada na Tabela D.1.

Tabela D.1 – Passo completo.

Nº de passo	B3	B2	B1	B0	Decimal
1	1	0	0	0	8
2	0	1	0	0	4
3	0	0	1	0	2
4	0	0	0	1	1

Fonte: Santos Filho (2012)

A sequência para energizar as bobinas, para realizar meio passo é apresentada na Tabela D.2.

Tabela D.2 – Meio passo.

Nº de passo	B3	B2	B1	B0	Decimal
1	1	0	0	0	8
2	1	1	0	0	12
3	0	1	0	0	4
4	0	1	1	0	6
5	0	0	1	0	2
6	0	0	1	1	3
7	0	0	0	1	1
8	1	0	0	1	9

Fonte: Santos Filho (2012)

Na Figura D.2, apresenta-se um exemplo de motor de passo, que foi utilizado em laboratório para testes com o STIM.

Figura D.2 – Foto do motor de passo utilizado em laboratório.



Fonte: Santos Filho (2012)

D.3 Ventiladores

As características dos ventiladores utilizados neste trabalho é descrita como:

- **Tensão de operação:** 12V.
- **Corrente:** 0,25A.
- **Diametro:** 120x120x25mm.

D.4 Motor CC

As características do motor CC utilizado neste trabalho é descrita como:

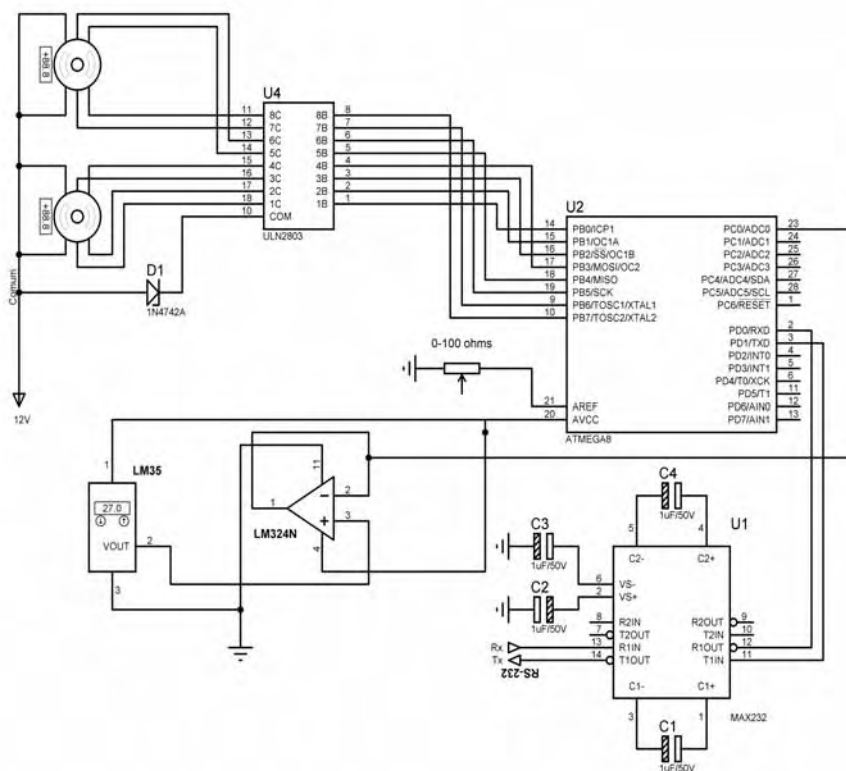
- **Tensão de operação:** 12V.
- **Corrente:** 1,5A.
- **Diâmetro:** 37mm.
- **Comprimento:** 57mm a 65mm.
- **Diâmetro do eixo:** 3mm.
- **Rotação:** 9500 rpm.

Apêndice E

Circuitos

E.1 Circuito STIM_1

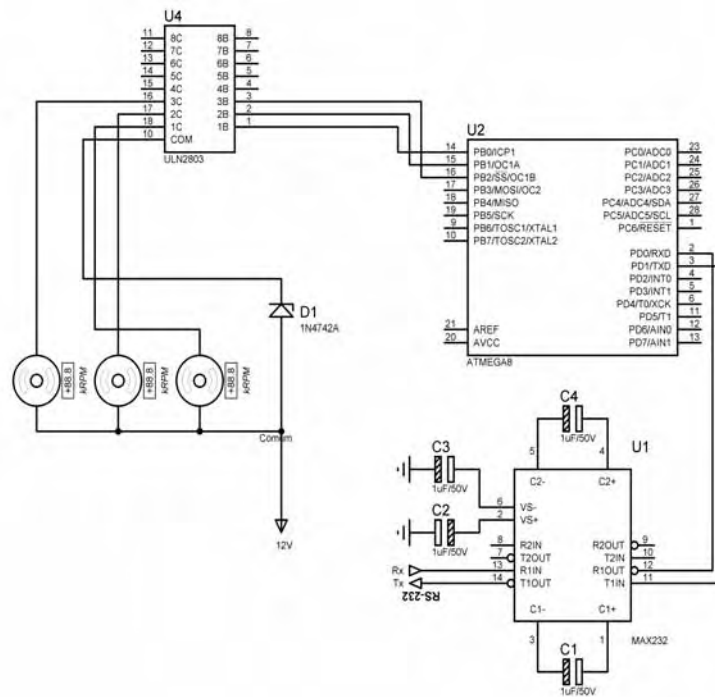
Figura E.1 – Circuito STIM_1 desenvolvido em laboratório.



Fonte: Santos Filho (2012)

E.2 Circuito STIM_2

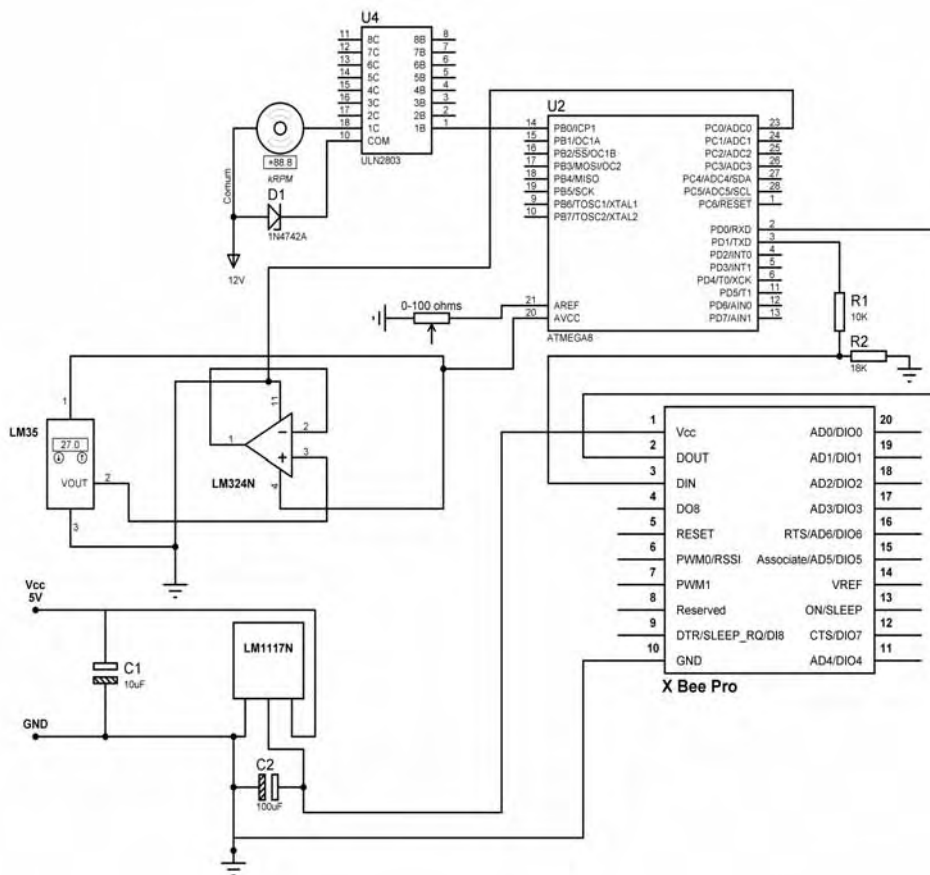
Figura E.2 – Circuito STIM_2 desenvolvido em laboratório.



Fonte: Santos Filho (2012)

E.3 Circuito WTIM_1

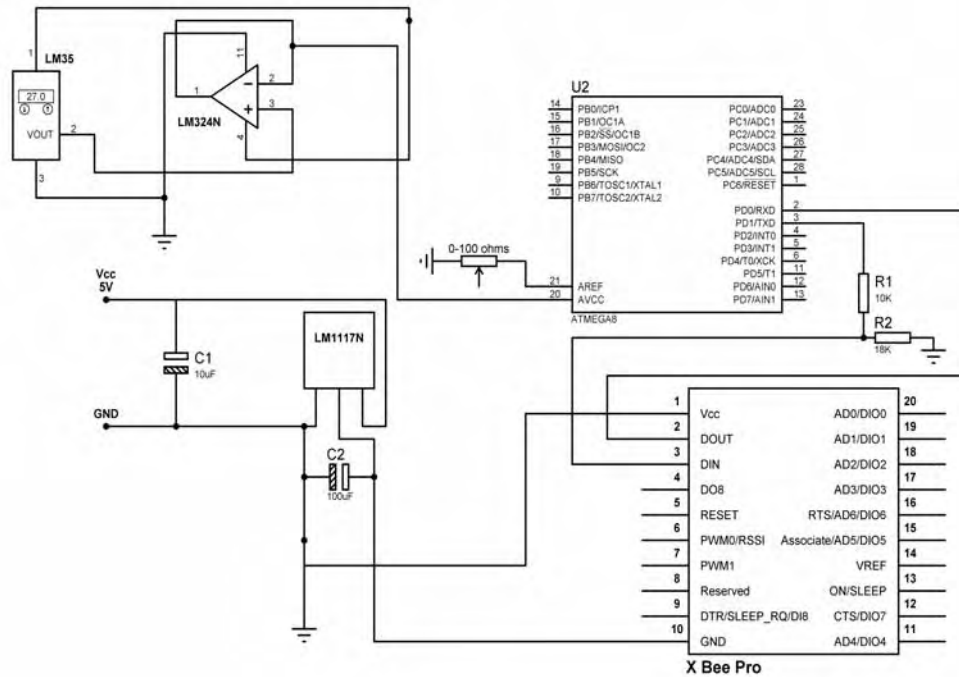
Figura E.3 – Circuito WTIM_1 desenvolvido em laboratório.



Fonte: Santos Filho (2012)

E.4 Circuito WTIM_2

Figura E.4 – Circuito WTIM_2 desenvolvido em laboratório.



Fonte: Santos Filho (2012)

Apêndice **F**

Configurações do Módulo XBee-Pro

Nesta seção, apresenta-se o módulo XBee/XBee-Pro utilizado no desenvolvimento do projeto e o módulo CON-USBee utilizado para a configuração dos módulos XBee-Pro. O XBee Pro é um módulo de RF que utiliza como comunicação com estações de trabalho a lógica da porta serial assíncrona. Através da porta serial, o módulo XBee-Pro pode realizar a comunicação com diferentes interfaces, sendo: USB e RS-485/232/422. Na Figura F.1, apresenta-se o XBee-Pro utilizado para realizar a comunicação entre o NCAP e o WTIM.

Figura F.1 – Módulo XBee-Pro.



Fonte: Pymble (2012)

Para o suporte e a configuração do XBee-Pro, foi utilizado o CON-USBee desenvolvido por Rogercon, para facilitar a conexão com os microcomputadores. O módulo CON-USBee oferece suporte para XBee/XBee-Pro, utiliza um conversor USB/Serial, regulador de tensão, comparador de tensão conectados aos LEDs (RSSI) que simulam a força do sinal de RF, LEDs indicadores de Tx e Rx, módulo ligado (ASS) e botão de “Reset”. Na Figura F.2 apresenta-se o módulo CON-USBee.

Na Figura F.3, apresenta-se o módulo XBee-Pro conectado ao microcomputador, utilizando o CON-USBee.

Para configurar o XBee-Pro, é utilizado o ambiente X-CTU, no qual a primeira interface usuário apresenta as interfaces de comunicação disponíveis para a comunicação serial. O XBee-Pro foi configurado na porta serial denominada COM3, de acordo com a

Figura F.2 – Módulo CON-USBees.



Fonte: Santos Filho (2012)

Figura F.3 – Módulo CON-USBees com XBee-Pro conectado ao microcomputador.



Fonte: Santos Filho (2012)

Figura F.4.

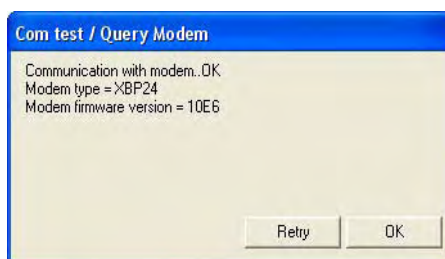
Figura F.4 – Interface usuário apresentada pelo ambiente X-CTU.



Fonte: Santos Filho (2012)

Para realizar o teste de comunicação entre o ambiente X-CTU e o módulo CON-USBees, é utilizada a opção “**Test/Query**”, apresentada na Figura F.4. Ao selecionar a opção “**Test/Query**”, são apresentadas para o usuário as configurações do módulo conectado no microcomputador de acordo com a Figura F.5.

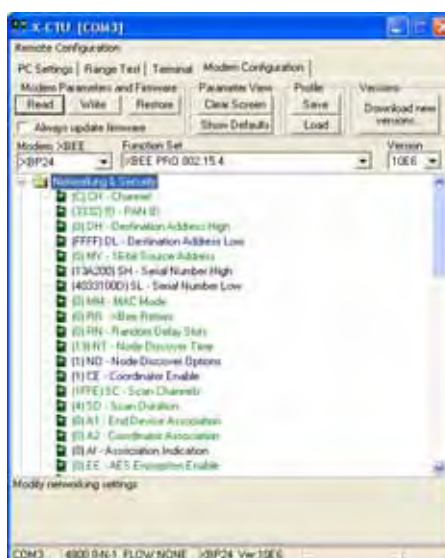
Figura F.5 – Teste de comunicação com módulo CON-USBee.



Fonte: Santos Filho (2012)

Para configurar os módulos XBee-Pro é utilizada a opção “Modem Configuration”. Na Figura F.6 apresenta-se a interface de configuração do módulo XBee-Pro e suas características.

Figura F.6 – Interface usuário de configuração do módulo XBee-Pro.

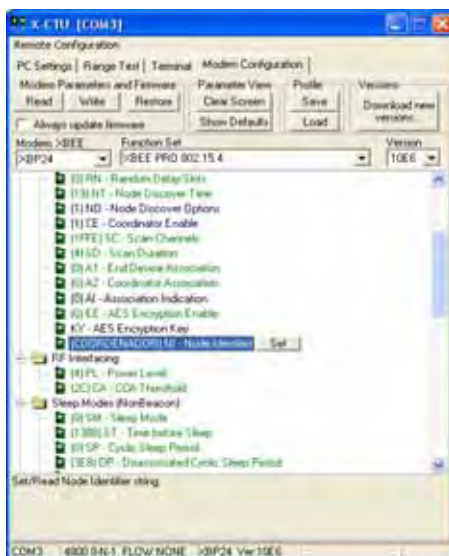


Fonte: Santos Filho (2012)

Neste trabalho, foram configurados os nomes do módulo XBee-Pro e a configuração da interface serial UART. Para facilitar a identificação na rede, foram definidos nomes para cada módulo XBee-Pro, sendo: Coordenador, WTIM_1 e WTIM_2. Na Figura F.7, apresentam-se a interface usuário e a opção “**Node Identifier**” para definir o nome do módulo XBee-Pro.

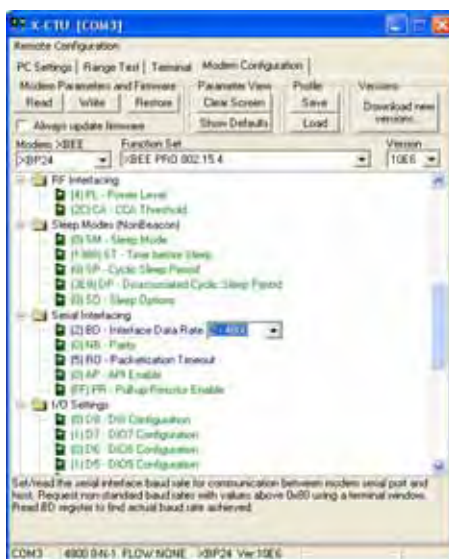
A interface serial foi configurada de acordo com a velocidade de transmissão definida no microcontrolador ATmega8 e no kit DE2. Neste trabalho, foram utilizados 4800 bps para o NCAP e os TIMs, deste modo, os módulos XBee-Pro foram definidos com a mesma velocidade de transmissão. Na Figura F.8 apresenta-se a interface usuário de configuração da velocidade de transmissão do módulo XBee-Pro.

Figura F.7 – Definição do nome do módulo XBee-Pro.



Fonte: Santos Filho (2012)

Figura F.8 – Configuração da velocidade de transmissão do módulo XBee-Pro.

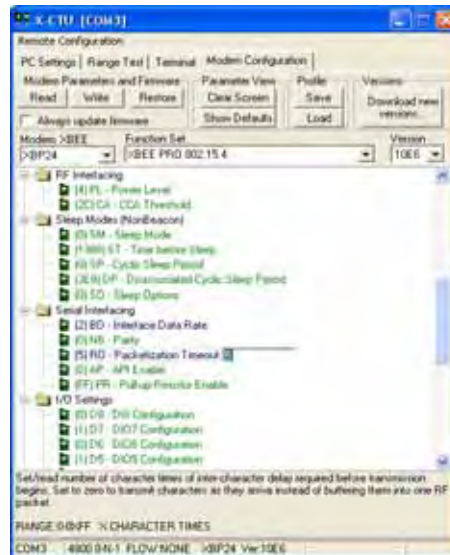


Fonte: Santos Filho (2012)

Um outro aspecto que deve ser ressaltado foi a opção “**Packetization Timeout**” em que realiza o armazenamento dos caracteres até chegar ao valor configurado pelo projetista, antes de enviar através da interface sem fio.

Na Figura F.9, apresenta-se a interface usuário de configuração da transmissão dos pacotes.

Figura F.9 – Configuração do buffer de caracteres.



Fonte: Santos Filho (2012)