LUÍS FILIPE F. B. S. ROSSI

SISTEMA PARA SENSORIAMENTO E CONTROLE PARA APLICAÇÕES EM BIOMECATRÔNICA

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Ciências.

São Paulo
2012

LUÍS FILIPE F. B. S. ROSSI

SISTEMA PARA SENSORIAMENTO E CONTROLE PARA APLICAÇÕES EM BIOMECATRÔNICA

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Ciências.

Área de Concentração:

Microeletrônica

Orientador:

Prof. Dr. Francisco Javier Ramirez Fernandez

São Paulo
2012

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, de março de 2012.

Assinatura do autor _____

Assinatura do orientador _____

FICHA CATALOGRÁFICA

Rossi, Luís Filipe Fragoso de Barros e Silva
Sistema para sensoriamento e controle para aplicações em biomecatrônica / L.F.F.B.S. Rossi. -- ed.rev. -- São Paulo, 2012.
113 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

1. Robótica 2. Controle digital 3. Sistemas distribuídos I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II. t.

Aos meus pais e à minha irmã.

AGRADECIMENTOS

À minha família, pelo apoio em todos os momentos difíceis.

À minha namorada, por aceitar trilhar este caminho ao meu lado.

Ao Professor Doutor Francisco Javier Ramirez Fernandez, meu orientador, pela confiança, paciência e ajuda prestada.

Ao professor Doutor Arturo Forner Cordero, por toda ajuda e auxílio no decorrer deste trabalho.

Ao grupo de Sensores Integráveis e Microssistemas (SIM), por todo apoio e ajuda nas dificuldades encontradas.

Ao grupo do Laboratório de Biomecatrônica, pelas conversas e sugestões.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pelo apoio financeiro concedido por meio de bolsa de mestrado.

À Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), pelo auxílio fornecido ao Laboratório de Biomecatrônica.

RESUMO

Diversos trabalhos relacionados ao desenvolvimento de dispositivos robóticos biomecatrônicos estão sendo realizados em vários laboratórios no mundo. Apesar desta crescente tendência, devido a uma falta de padronização nas tecnologias utilizadas, em especial no sistema de sensoriamento e controle, há uma grande divergência nos sistemas resultantes. De forma a se conseguir atender os requisitos dos projetos, muito tempo é despendido no desenvolvimento de sistemas de sensoriamento e controle dedicados. Dentro deste cenário, neste trabalho foi projetado e implementado um sistema de sensoriamento e controle modular específico para sistemas robóticos. Este foi desenvolvido de forma a poder ser utilizado em diversos projetos reduzindo o esforço para a sua implementação. O referido sistema foi dividido em três módulos: Processador Central, Nós e Rede de Comunicação. Foi dada uma especial atenção no aspecto relacionado à comunicação por ser um fator-chave para se conseguir manter compatibilidade entre diferentes sistemas. Uma rede de comunicação denominada R-Bone foi desenvolvida pelo fato de que os sistemas existentes não atendem aos requisitos propostos. Uma descrição conceitual do sistema projetado é apresentada e a sua implementação detalhada. Todos os aspectos técnicos relevantes foram descritos de forma a facilitar a sua replicação por outros grupos. Um driver para sistema operacional Linux foi desenvolvido em conjunto com uma camada de abstração para simplificar o seu uso. Os testes realizados demonstraram que o sistema desenvolvido atende os requisitos propostos, mantendo uma condição de estabilidade adequada em seu tempo de resposta, baixa latência e pouca defasagem entre os sinais coletados pelos sensores. De forma a contribuir para uma possível padronização dos sistemas utilizados na área, todos os arquivos e informações relevantes para a replicação do sistema proposto foram disponibilizados sob a licença GNU LGPL em um servidor SVN.

Palavras-chave: biomecatrônica, FPGA, sistema modular, controle distribuído, robótica.

ABSTRACT

Several works related to the development of biomechatronic robotic systems are being taken in several laboratories around the world. Despite this increasing trend, due to a lack of standardization in the used technologies, in special related to the control and sensing system, there is a wide divergence in the resulting system. In order to meet the project requirements, a lot of time is spent in the development of a custom control and sensing system. In this scenario, a modular sensing and control system specifically designed to be used in robotic systems, was designed and implemented. The last was developed in order to be used in several projects, thus reducing the effort spent on its implementation. This system was divided into three modules: Central Processor, Nodes and Communication Network. A special attention was given to the aspects related to the communication as it is the key-factor to keep compatibility among different systems. A communication network named R-Bone was developed, and its implementation was detailed. All the relevant technical aspects were described in order to facilitate its replication by other groups. A driver for the Linux operating system was developed in conjunction with an abstraction layer to simplify its use. The tests demonstrated that the system meets the proposed requirements, keeping a proper stability condition in the response time, low latency and little skew between the signals collected by the sensors. In order to contribute to a possible standardization of the systems used in the biomechatronics field, all the files with relevant information to make possible the replication of the proposed system were made available under the GNU LGPL license in a SVN server.

Key-words: biomechatronics, FPGA, modular system, distributed control, robotics.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADC	*Analog-to-digital Converter*
API	*Application Programming Interface*
ASIC	*Application-specific Integrated Circuit*
ASSP	*Application-specific Standard product*
BGA	*Ball Grid Array*
CAN	*Controller Area Network*
CPU	*Central Processing Unit*
CRC	*Cyclic Redundancy Check*
DAC	*Digital-to-analog Converter*
DARPA	*Defense Advanced Research Projects Agency*
DC	*Direct Current*
DDR	*Double Data Rate*
DMA	*Direct Memory Access*
DOF	*Degrees of Freedom*
DSC	*Digital Signal Controller*
DSP	*Digital Signal Processor*
EEG	Eletroencefalografia
EMG	Eletromiografia
FIFO	*First In First Out*

FPGA	*Field-programmable Gate Array*
FPU	*Floating-point Unit*
FSM	*Finite State Machine*
HRP	*Humanoid Robotics Project*
I2C	*Inter-Integrated Circuit*
IEEE	*Institute of Electrical and Electronic Engineers*
ISA	*Industry Standard Architecture*
LCD	*Liquid Crystal Display*
LGPL	*Lesser General Public License*
LSB	*Least Significant Bit*
LUT	*Lookup Table*
LVDS	*Low-voltage Differential Signaling*
LVTTL	*Low Voltage Transistor-Transistor Logic*
MAC	*Multiply-accumulate*
M-LVDS	*Multipoint Low-voltage Differential Signaling*
MSB	*Most Significant Bit*
PCI	*Peripheral Component Interconnect*
PLL	*Phase-locked Loop*
PWM	*Pulse-width Modulation*
RAM	*Random-access Memory*
SPI	*Serial Peripheral Interface*
SVN	*Subversion*

TTL	*Transistor-Transistor Logic*
UART	*Universal Asynchronous Receiver/Transmitter*
USB	*Universal Serial Bus*
USP	Universidade de São Paulo
μC	Microcontrolador

1 INTRODUÇÃO

O presente trabalho foi desenvolvido no âmbito de uma parceria entre o Laboratório de Microeletrônica e o Laboratório de Biomecatrônica da Escola Politécnica da Universidade de São Paulo. Este último, por sua vez, é um dos poucos grupos a nível nacional que realiza estudos e desenvolvimento na campo de biomecatrônica. O grupo em questão tem como objetivos tanto o estudo e modelagem do sistema de controle motor humano, como o desenvolvimento de dispositivos robóticos bioinspirados. Atualmente existem dois projetos nos quais o presente trabalho está incluído, sendo um focado no desenvolvimento de um exoesqueleto para membros superiores (FORNER-CORDERO et al., 2011) e outro no desenvolvimento de um exoesqueleto de membros inferiores.

A seguir será realizado um resumo do estado da arte relacionado à biomecatrônica, para então definir o problema que este trabalho se propôs a mitigar. Visto que a robótica industrial se encontra em um cenário muito diferente do aqui abordado, pois já possui um mercado muito bem consolidado, sempre que forem citados os termos robótica e sistemas robóticos, estarão sendo excluídos destes grupos os robôs industriais. É importante ressaltar que a presente revisão bibliográfica organizou a descrição dos dispositivos de acordo com o grupo de pesquisa que os desenvolveu. Esta forma de abordagem foi escolhida para evidenciar os argumentos utilizados na Seção 1.2 deste capítulo.

1.1 Biomecatrônica e Robótica

Dentro do campo de desenvolvimento robótico na área da biomecatrônica, existem basicamente duas grandes vertentes. A primeira se dedica ao desenvolvimento de exoesqueletos robóticos, cujos resultados podem ser usados tanto para fins de pesquisa, quanto para aplicações comerciais, incluindo assistência motora e reabilitação. A segunda vertente, por sua vez, se dedica ao desenvolvimento de robôs bioinspirados e dentro deste grupo se destacam os robôs humanoides, os quais tem diversas características interessantes devido à sua cinemática compatível com a dos seres humanos.

1.1.1 Exoesqueletos

O primeiro exoesqueleto, denominado Hardiman, foi desenvolvido pela General Electric (MAKINSON, 1971). Devido à complexidade de manuseio e insegurança ele foi impossibilitado de se locomover (ZOSS; CHU, 2006). Alguns anos depois, outro sistema foi desenvolvido pelo Instituto Mihajlo Pupin (VUKOBRATOVIC; HRISTIC; STOJILJKOVIC, 1974) o qual se resumia basicamente a um equipamento de assistência motora para membros inferiores. Porém este último era limitado a seguir movimentos pré-programados. Ambos os dispositivos eram alimentados por fonte de energia estacionária (ZOSS; CHU, 2006).

Mais recentemente, com os avanços tecnológicos, aumento de capacidade de processamento e atuadores mais eficientes, diversos laboratórios no mundo estão pesquisando e desenvolvendo exoesqueletos.

Na Universidade de Tsukuba, no Japão, o Professor Sankay apresentou o exoesqueleto HAL 3 no ano de 2000, o qual consiste de um exoesqueleto de membros inferiores. Ele é acionado por motores elétricos e controlado por um computador que tenta prever o movimento ou a intenção do usuário

através de sensores que monitoram sinais de EMG e sensores de força posicionados na sola do exoesqueleto (LEE; SANKAI, 2002a; LEE; SANKAI, 2002b). Mais recentemente, foi apresentado por Sankay o exoesqueleto HAL 5, porém pouca informação relacionada ao seu funcionamento foi divulgada. O seu principal foco de aplicação é auxiliar idosos e pessoas com musculatura fraca a carregar objetos pesados e se locomover, além de auxiliar na reabilitação de pacientes com lesões na medula espinhal (SANKAI, 2006).

Na Universidade da Califórnia, nos Estados Unidos, foi desenvolvido o BLEEX, financiado pela *Defense Advanced Research Projects Agency* (DARPA). Ele consiste de um exoesqueleto para membros inferiores que tem como objetivo auxiliar na carga de equipamentos pesados. Ele é acionado por atuadores lineares hidráulicos (ZOSS; KAZEROONI; CHU, 2005) e foi desenvolvido um sistema chamado de ExoNet para realizar o seu controle e sensoriamento de uma forma distribuída (KIM; KAZEROONI, 2004; ZOSS; CHU, 2006; CHU; KAZEROONI; ZOSS, 2005; STEGER; KIM; KAZEROONI, 2006; KAZEROONI et al., 2005). Novas versões mais aperfeiçoadas, como o HULC™ e o ExoHiker™, foram posteriormente desenvolvidas, porém pouca informação técnica foi divulgada.

No *Florida Intitute for Human and Machine Cognition* (IHMC) foi desenvolvido um exoesqueleto de membros inferiores utilizando atuadores elásticos em série rotativos. O controle principal é realizado por um computador externo ao exoesqueleto e um computador com um formato PC/104, com placas de expansão customizadas, é utilizado como um sistema intermediário. O exoesqueleto não possui nenhuma forma de alimentação elétrica embarcada, sendo ela feita por cabos (KWA et al., 2009).

No Brasil existem algumas iniciativas recentes além do Laboratório de Biomecatrônica da Escola Politécnica da USP. Em 2010, foi apresentado um

exoesqueleto de membro superior desenvolvido por Renato Varoto, na Escola de Engenharia de São Carlos da Universidade de São Paulo, sob a orientação do Prof. Dr. Alberto Cliquet Junior. O referido exoesqueleto foi projetado tanto para auxiliar no processo de reabilitação de pacientes com lesão medular, quanto para auxiliar o paciente em tarefas diárias. O controle principal é realizado por voz e o sistema eletrônico para a sua implementação foi construído dedicado ao projeto (VAROTO, 2010; VAROTO; BARBARINI; CLIQUET, 2008).

Também na Escola de Engenharia de São Carlos da USP, está sendo desenvolvido um exoesqueleto de membros inferiores baseado em atuadores elásticos em série. No âmbito desse projeto já foi desenvolvido, por Bruno Jardim, uma órtese ativa para tornozelo e pé, orientado pelo Prof. Dr. Adriano Siqueira (JARDIM; SIQUEIRA, 2009). O controle é realizado através de um driver de potência EPOS da Maxon Motors em conjunto com um computador estacionário (JARDIM, 2009).

1.1.2 Robôs Humanoides

O primeiro robô humanoide bípede que se tem conhecimento, foi o WABOT-1 desenvolvido na Universidade de Waseda, no Japão (KAJITA; ESPIAUR, 2008). Assim como com os exoesqueletos robóticos, devido aos recentes avanços tecnológicos, houve um grande crescimento no número de trabalhos relacionados à área de robôs humanoides.

ASIMO foi um dos responsáveis por desencadear um crescimento significativo do número de trabalhos na área de robôs humanoides bípedes (KANEKO et al., 2008). A sua primeira versão foi demonstrada em 2000 (SAKAGAMI et al., 2002), porém em 2005 um novo robô com algumas melhorias foi apresentado. Este último tem um total de 34 graus de liberdade e, de forma a poder se movimentar com mais flexibilidade, faz o uso de um sistema de controle

proprietário chamado de *I-Walk*, podendo assim modificar a direção de sua caminhada continuamente (HONDA, s.d.).

Outro robô, denominado HRP-1, foi desenvolvido pela HONDA no âmbito do *Humanoid Robotics Project* (HRP), financiado pelo *Ministry of Economy, Trade and Industry* (METI) do Japão (HIRUKAWA et al., 2004). Devido à uma série de limitações do HRP-1, o *National Institute of Advanced Industrial Science and Technology* (AIST) reformulou o projeto e desenvolveu o HRP-1S. Este, por sua vez, é baseado em uma arquitetura de controle e comunicação centralizada e possuí um total de 30 DOFs. O processamento principal do robô é baseado no computador VMIVME-7740-877, fabricado pela VMIC. Toda aquisição de sinais e atuação é efetuada através de uma placa proprietária desenvolvida pela HONDA (YOKOI et al., 2004).

Além dos sistemas citados, uma série de robôs foram desenvolvidos em uma parceria do AIST com a *Kawada Industries*. O primeiro foi o HRP-2P, o qual foi implementado ainda no âmbito do *Humanoid Robotics Project*. Com um total de 30 DOFs, ele foi projetado como uma plataforma de desenvolvimento de robôs para serem utilizados como ajudantes de seres humanos. Ainda baseado em um sistema de sensoriamento e controle totalmente centralizado, foram desenvolvidas placas customizadas para serem conectadas ao barramento PCI de forma a reduzir a quantidade de componentes no sistema. No entanto, o seu processamento foi distribuído entre duas CPUs, de forma a separar as rotinas de controle do processamento da visão e do áudio (KANEKO et al., 2002).

O HRP-2, por sua vez, foi desenvolvido baseado nos conhecimentos adquiridos com o HRP-2P. A parte mecânica foi aperfeiçoada de forma a reduzir alguns problemas de superaquecimento observados em seu predecessor. O sistema de controle manteve uma arquitetura similar, com pequenas adapta-

ções de forma a aumentar a imunidade a ruídos (KANEKO et al., 2004).

Apresentado em 2005, o HRP-3P foi desenvolvido com diversas melhorias significativas em seu sistema, sendo um dos principais focos do projeto o aumento da confiabilidade do robô como um todo. Com 36 DOFs, a sua estrutura mecânica foi projetada de forma a ser protegida contra água e poeira. O sistema de sensoriamento e controle foi totalmente reformulado, pois notaram-se uma série de problemas devido ao cabeamento excessivo de seu predecessor, o HRP-2 (AKACHI et al., 2005). Um sistema de comunicação proprietário baseado em Ethernet foi desenvolvido de modo a possibilitar ciclos de controle à frequências na ordem de alguns KHz. De forma a conseguir satisfazer os requisitos de temporização do sistema, um protocolo customizado foi implementado reformulando assim a topologia tradicional da Ethernet, transformando-a em um sistema de comunicação em anel. De forma a criar um sistema de controle mais elaborado, uma série de placas com um processador ARM9 foram distribuídas pelo corpo do robô para atuarem como processadores locais, criando assim um sistema distribuído (KANEHIRO et al., 2006).

O seu sucessor, nomeado HRP-3, teve o sistema mecânico reformulado obtendo um total de 42 DOFs. De forma a possibilitar o robô a manusear alguns objetos, uma nova mão articulada foi projetada. O sistema de comunicação desenvolvido foi substituído por diversos canais de comunicação CAN, porém a estrutura de sensoriamento e controle distribuída foi mantida. Esta alteração foi feita de forma a aumentar a confiabilidade e simplificar a manutenção do sistema (KANEKO et al., 2008).

Em 2009 foi apresentado um robô com a aparência de uma mulher japonesa denominado HRP-4C. Este foi projetado de forma a ter a capacidade de andar e atuar como um ser humano. O mesmo manteve um total de 42 DOFs,

porém a mão foi reformulada de forma a mitigar alguns problemas notados no HRP-3. O sistema de sensoriamento e controle é baseado em 10 canais de comunicação CAN em conjunto com placas de controle customizadas distribuídas. O processamento é realizado em uma placa com o formato PC/104 com um processador Pentium® M. Diversas adaptações mecânicas foram realizadas de forma a conseguir se aproximar da aparência de um ser humano (KANEKO et al., 2009; KANEKO et al., 2011a).

Atualmente o HRP-4 é o robô mais avançado desenvolvido pela *Kawada Industries*. O seu sistema mecânico e de controle é similar ao HRP-4C diferenciando-se principalmente pelo fato de ter apenas 34 DOFs. Diversas modificações foram realizadas comparados aos seus predecessores de forma a reduzir o seu peso e custo de fabricação. Estima-se que o seu custo é 30% inferior ao do HRP-2 (KANEKO et al., 2011b).

O German Aerospace Center (DLR) desenvolveu diversos dispositivos robóticos bioinspirados. Justin é um robô humanoide que se movimenta com uma plataforma motorizada por rodas. Ele emprega um conjunto de braços, também desenvolvido pelo DLR, extremamente leve e versátil (HIRZINGER et al., 2002). Ele faz uso de um sistema de sensoriamento e controle distribuído baseado em diversos padrões de comunicação como EtherCAT, SERCOS, CAN e SpaceWire. Ele faz uso de uma arquitetura não convencional, onde um coletor de dados atua como um mestre EtherCAT. Este, por sua vez, coleta todos os dados e os envia para o sistema de processamento principal, o qual é um escravo na rede. Apesar da simplicidade de operação do sistema de processamento principal, em (FUCHS et al., 2009) foi comentado haver uma certa complexidade na configuração do coletor de dados. O seu sistema de processamento principal é formado por quatro computadores com o formato MiniITX munidos de um processador *Dual Core* e estão interconectados utili-

zando conexões Gigabit Ethernet a um switch.

Além do Justin, o DLR desenvolveu duas mãos robóticas em parceria com o *Harbin Institute of Technology* (HIT). A mais proeminente e avançada delas é a DLR/HIT *Hand* II. Com um total de 15 DOFs distribuídos em uma palma e cinco dedos articulados, é considerada umas das mãos robóticas mais complexas da atualidade. O controle de cada dedo é realizado por um DSP em conjunto com um FPGA. Este, por sua vez também implementa a comunicação com o sistema de controle da palma utilizando um meio físico baseado em M-LVDS e um protocolo de comunicação proprietário chamado de PPSeCo. O controlador da palma também é baseado em um FPGA o qual implementa dois processadores NIOS II da Altera para realizar o gerenciamento de sua lógica e comunicação. A palma utiliza Ethernet, CAN e PPSeCo como interface de comunicação. Ela, por sua vez, é controlada por uma placa customizada que se comunica com um computador pelo barramento PCI. Nesta placa se encontram um DSP TMS320C6713 da Texas Instruments e um FPGA para realizar a comunicação (LIU et al., 2008).

Para finalizar o DLR recentemente apresentou um novo sistema de mão e braço robóticos, cujo sistema de sensoriamento e controle também é baseado em uma arquitetura distribuída. Este, por sua vez, utiliza tanto SpaceWire como BiSS para realizar a comunicação de dados com os módulos de controle distribuídos. A comunicação é implementada utilizando os FPGAs XC3S500EP132 e V5LX50 da Xilinx® (GREBENSTEIN et al., 2011).

Na *Technical University Munich* (TUM), foi desenvolvido o robô LOLA com objetivo de estudar o controle da marcha em robôs bípedes. Ele tem um total de 25 DOFs, sendo que 14 deles estão distribuídos em suas duas pernas. O robô faz uso de um sistema de sensoriamento e controle distribuído baseado em SERCOS-III para realizar a transmissão de dados. Todos os módulos de

processamento distribuído são customizados. Eles são implementados utilizando o DSP MC56F8367 da Freescale como processador local e um FPGA XC3S400 da Xilinx® para implementar a comunicação de rede (LOHMEIER; BUSCHMANN; ULBRICH, 2009). Porém, foi relatado que tal sistema baseado em SERCOS-III ainda não se encontra em operação e como substituto estão sendo utilizados oito canais de comunicação CAN realizando comunicação ponto-a-ponto com os módulos (LOHMEIER, 2010).

O robô humanoide Albert HUBO foi desenvolvido pelo *Korea Advanced Institute of Science and Technology* (KAIST) e se destaca pelo seu rosto multiarticulado no formato de Albert Einstein. A cabeça do robô incorpora um total de 31 servomotores enquanto o resto do corpo é atuado por um total de 35 motores DC. Ele é baseado em um sistema de controle distribuído que utiliza CAN para realizar a comunicação entre um processador central e diversos *Micro Processor Units* (MPU) distribuídos pelo sistema. O processamento principal foi implementado utilizando a placa PCM-3370 da Advantech, a qual possui processador Pentium® III capaz de processar informações a 900 Mhz (OH et al., 2006).

Na *Waseda University*, no Japão, foi desenvolvido o Wabian II, o qual é um robô humanoide bípede projetado para testar equipamentos de assistência motora. Com 41 DOFs, grande parte do seu sistema de controle é baseado em módulos comerciais. Este, por sua vez, é totalmente centralizado e utiliza três HRP *Interface board* para realizar o controle dos drivers de motores e aquisição dos sinais dos sensores. Elas se comunicam com o computador principal, cujo processador é um Pentium® M, através do barramento PCI. Os drivers dos motores são fabricados pela empresa japonesa Tokushu Denso (OGURA et al., 2006).

ROTTO foi desenvolvido pela *Otto-von-Guericke-University Magdeburg*,

na Alemanha. Ele consiste de um robô bípede de membros inferiores com um total de 12 DOFs. Além disso, se movimenta utilizando atuadores elásticos em série customizados e o sistema de controle é distribuído com a comunicação feita via EtherCAT. Para tal foram utilizados os processadores netX, os quais são baseados na arquitetura ARM e possuem um controlador de rede multiprotocolo. Estes também são responsáveis por efetuar o controle dos atuadores localmente, atuando como drivers dos motores. O sistema de processamento central consiste de um computador convencional o qual não se encontra embarcado no robô (KONYEV et al., 2009; MELNYKOV et al., 2010).

O *Institute of Human and Machine Cognition* (IHMC) desenvolveu o M2V2 para realizar estudos sobre a marcha em robôs bípedes. Este, por sua vez, consiste de um robô bípede de membros inferiores também acionado utilizando atuadores elásticos em série. Seu sistema de sensoriamento e controle é centralizado e é implementado utilizando uma série de placas com o formato PC/104 empilhadas (PRATT; KRUPP, 2008).

1.2 Caracterização do Problema

Apesar de diversas universidades e centros de pesquisa estarem desenvolvendo dispositivos robóticos para aplicações na área da biomecatrônica, não há uma padronização na arquitetura empregada, situação que promove uma grande divergência nos sistemas resultantes.

Alguns projetos, como o Wabian II e o M2V2, utilizam uma arquitetura centralizada de modo a utilizar módulos disponíveis comercialmente. Porém, grande parte dos trabalhos recentes vêm sendo baseados em sistemas distribuídos e consequentemente em comunicação digital.

Grande parte dos projetos com um requisito de taxa de transmissão razo-

avelmente baixo são baseados em CAN ou outro *fieldbus* equivalente. Alguns trabalhos recentes, como o (YU et al., 2007), se dedicaram ao estudo e implementação uma rede de controle baseada em CAN para um robô humanoide bípede.

Sistemas com requisitos maiores e mais sofisticados divergem bastante na rede de comunicação escolhida. Alguns, como ROTTO e LOLA, fazem uso de variações comerciais da Ethernet para aplicações *Real-Time*. Outros, como no caso do HRP-3P, desenvolvem uma variação de Ethernet customizada.

Recentemente alguns trabalhos focados no estudo de sistemas de sensoriamento e controle vêm utilizando IEEE 1394 como o sistema de comunicação implementado (THIENPHRAPA; KAZANZIDES, 2011; KAZANZIDES; THIENPHRAPA, 2008; SARKER et al., 2006). Com base nos resultados observados na literatura é possível afirmar que grande parte dos trabalhos tenta adaptar as tecnologias existentes que foram projetadas para outros fins. Um número bem mais restrito de projetos desenvolvem uma rede totalmente dedicada, como no caso do ExoNet do projeto BLEEX.

Adicionalmente a esta divergência, apesar de uma pequena descrição dos sistemas de controle implementados, devido a razões comerciais, a grande maioria dos grupos não provê um fácil acesso a detalhes relacionados à arquitetura desenvolvida. Esta prática leva à necessidade de uma constante "reinvenção da roda", mesmo para projetos similares, resultando em abordagens diferentes e incompatíveis.

Como comentado por Fujita (FUJITA; KAGEYAMA, 1997), uma padronização cria um ambiente de desenvolvimento muito mais eficiente, já que permite o uso de componentes desenvolvidos por empresas e outros grupos. Algumas iniciativas, como o projeto Orocos (BRUYNINCKX, 2001), obtiveram sucesso em criar um *middleware* padronizado para sistemas robóticos, possibilitando

assim uma eficiente troca de códigos desenvolvidos por diferentes grupos ao redor do mundo.

No nível de hardware uma padronização pode trazer benefícios em diversas escalas. Em uma visão mais macroscópica pode estimular o desenvolvimento comercial de produtos dedicados para sistemas robóticos, assim evitando o tempo atualmente gasto com o desenvolvimento de sistemas dedicados. Além disso, possibilitaria a criação de ambientes de colaboração entre diferentes grupos, possibilitando assim cada um se especializar em certos aspectos científicos e tecnológicos, levando à uma maior eficiência do tempo empregado nos projetos.

Adicionalmente, exoesqueletos e robôs bípedes são sistemas custosos e muitas vezes é inviável a construção de diversas unidades. Além disso, muitas das arquiteturas implementadas foram projetadas para fins bens específicos, dificultando a utilização de um mesmo sistema para várias aplicações. Até então, pouco se falou na literatura sobre sistemas robóticos para múltiplos usos. Normalmente eles têm uma finalidade muito bem definida e qualquer grau de flexibilidade gerado é um benefício secundário decorrente de escolhas que objetivaram um ganho de performance. Uma possível modularização e padronização facilitaria a criação de sistemas robóticos com arquiteturas flexíveis, possibilitando assim a múltiplos projetos serem realizados com um único dispositivo.

Outro ponto importante é que muitos laboratórios costumam ter uma grande interação com profissionais das áreas de fisioterapia, ortopedia, educação física entre outras, que usualmente não têm conhecimento técnico em sistemas de controle. Além disso, laboratórios que realizam pesquisas na área de biomecatrônica muitas vezes têm um foco no desenvolvimento em algoritmos e os pesquisadores envolvidos podem não ter um conhecimento técnico

em eletrônica suficiente para realizar o projeto e construção de sistemas complexos. Desta forma, é interessante a existência de um sistema reutilizável em diversas aplicações diferentes e principalmente que o mesmo não sofra alterações em nível técnico muito drásticas, além de ter uma interface razoavelmente simples de forma a acelerar o ciclo de desenvolvimento.

Desta forma, fica evidente a importância da existência de um sistema de controle padronizado, que seja flexível, modular, de simples uso e ao mesmo tempo tenha um desempenho suficiente de forma a não limitar sua aplicabilidade. Vistas as novas iniciativas de pesquisa e desenvolvimento na área de biomecatrônica no Brasil e o fato de não existir nenhum sistema desenvolvido nacionalmente, ou disponível para uso que tenha tais características, se faz conveniente o projeto e desenvolvimento de um sistema voltado para tais aplicações, de forma a evitar os problemas citados anteriormente.

1.3 Objetivos

O objeto geral deste projeto é o desenvolvimento e implementação de um sistema de sensoriamento, processamento e controle para aplicações em biomecatrônica (e.g. Robôs Humanoides e Exoesqueletos).

O objetivo deste sistema é servir como base para diversas aplicações diferentes, sem haver a necessidade de realizar mudanças drásticas em seu hardware, desta forma servindo como uma ferramenta de pesquisa. Como principais requisitos ele deve ser flexível, modular, simples de ser replicado e principalmente deve criar uma camada de abstração, disponibilizando, de forma transparente e simples de ser implementada, uma interface para os periféricos utilizados na aplicação em questão.

Tal sistema deve ser concebido de forma que o usuário (pesquisador) não

tenha a necessidade de saber/aprender conhecimentos técnicos específicos relativos ao sistema em questão, salvo no caso da implementação de um periférico (sensor ou atuador) não previamente implementado. O único requisito para o seu uso deve ser o conhecimento, por um membro da equipe, de programação em C e um conhecimento básico em eletrônica.

Não devem existir limitações quanto à implementação de sensores e atuadores no sistema se o conjunto dos mesmos não ultrapassar os limites de taxa de comunicação e processamento definidos. Qualquer periférico, que siga as restrições de capacidade do sistema e tenha como interface o protocolo e meio físico compatíveis, deve poder ser implementado sem nenhum tipo de instalação ou desenvolvimento de softwares, salvo a implementação com camada de abstração previamente desenvolvida.

Além disso, de forma que vários projetos possam ser desenvolvidos em paralelo, trocas de sensores, atuadores e outros periféricos específicos de cada projeto devem ser de simples execução, envolvendo poucos conectores, de fácil manuseio e sem ter impacto significativo na arquitetura do sistema. Para finalizar, todas as partes utilizadas para a sua confecção devem ser de simples acesso, possibilitando a sua replicação por outros grupos.

1.4 Materiais e Métodos

A seguir serão apresentados os conceitos utilizados no projeto e implementação do sistema desenvolvido por este trabalho. Em seguida serão apresentadas as tecnologias utilizadas como base de forma a atender os objetivos e requisitos definidos.

1.4.1 Arquitetura do Sistema

Um sistema robótico basicamente consiste de uma lógica de controle, que utiliza dados amostrados de sensores, para tomar decisões, controlar atuadores ou passar alguma informação relevante para o operador. Tal lógica pode ser processada por um ou mais processadores, de uma forma centralizada ou distribuída.

De forma geral o sistema desenvolvido tem como objetivo o estudo de algoritmos e modelos de caráter centralizado, desta forma a existência de um Processador Central se faz conveniente. É importante ressaltar que a existência de um Processador Central não inviabiliza de nenhuma forma a existência de processadores locais ou auxiliares.

Como este trabalho visa desenvolver uma ferramenta sem uma implementação definida, nenhum sensor ou atuador será especificado. Sendo assim, estes serão tratados como Nós no sistema, os quais podem representar tanto sensores e atuadores diretamente ou um conjunto deles. Para finalizar, o Processador Central deve trocar informações com os Nós (sendo tal troca de forma analógica ou digital), assim sendo, deve existir uma rede de conexões entre eles. Na Figura 1 é demonstrado o sistema conceitual descrito.

Em vista da diferença entre os requisitos existentes em diferentes aplicações, é de extrema importância que tanto o Processador Central quanto Nós sejam de caráter modular. Porém, um dos pontos-chave para se conseguir criar alguma forma de padronização e modularizarão em arquiteturas de controle, é a padronização do sistema de comunicação. Isto possibilita uma grande flexibilidade na inclusão ou exclusão de sensores e atuadores sem alterar de uma forma expressiva a arquitetura do sistema. Desta forma, ficou definido que a rede de comunicação será de caráter fixo. Porém, tal definição

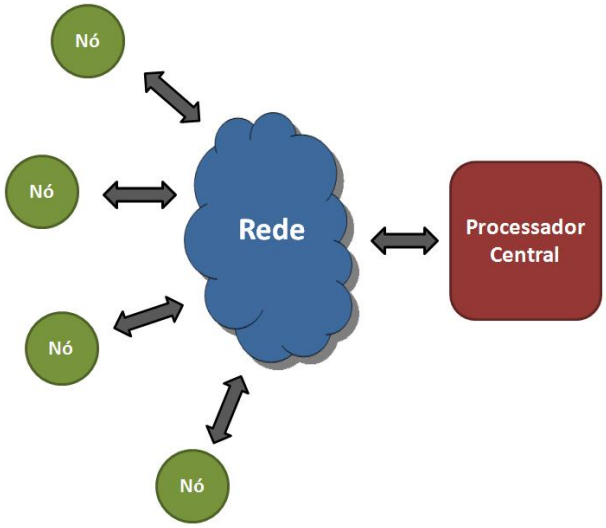


Figura 1: Sistema Conceitual

traz consequências diretas ao projeto como um todo. Caso a referida rede seja mal dimensionada e projetada, o sistema de controle desenvolvido ficará comprometido. Desta forma, uma grande parcela do esforço empregado neste trabalho foi relacionado à rede de comunicação. A seguir serão descritas algumas definições tomadas como base em seu projeto.

A existência de múltiplas interfaces conectadas diretamente ao Processador Central, sendo elas analógicas ou digitais, dificulta a implementação de qualquer sistema de controle, pois cada interface deve ser tratada separadamente. Desta forma, é conveniente que, na visão do Processador Central, todos os Nós estejam disponíveis através de uma única interface. Fica evidente então, que para tal, esta interface deve ser de caráter digital. Todas as interfaces com sinais analógicos devem sem implementadas dentro dos Nós, de modo a ficarem transparentes para o Processador Central.

Tendo estas definições, alguns requisitos foram definidos para serem considerados durante a fase de projeto.

(a) **Determinístico:** Como o sistema de comunicação desenvolvido será utilizado em sistemas robóticos que vão interagir diretamente com seres hu-

manos, atuar em atividades críticas, ou servir como dispositivos assistivos (como exoesqueletos), ele precisa atuar de uma forma determinística diante de erros de comunicação.

(b) **Baixa Latência:** Normalmente, sistemas robóticos têm como requisito realizar comunicações baseadas em pequenos pacotes de dados (10 a 100 bytes) transmitidos entre diversos Nós distribuídos no sistema e o Processador Central. Desta forma, de modo a possibilitar altas frequências de controle, que normalmente variam entre 1 a 10 Khz (KAZANZIDES; THIENPHRAPA, 2008), é necessário haver uma baixa latência entre duas mensagens consecutivas.

(c) **Simples de Replicar:** Alguns dos sistemas citados no início deste Capítulo utilizam uma rede de comunicação de alta performance, porém muitas vezes baseada em circuitos integrados difíceis de se adquirir, muito caros, ou complexos de se implementar, como no caso de FPGAs com encapsulamento BGA. Estas características implicam em uma dificuldade de replicação do sistema descrito, principalmente por diferentes grupos. Além disso, a dependência por circuitos integrados de difícil acesso ou muito específicos, cria uma dependência de alto risco, já que uma possível descontinuação dos mesmos implicaria em uma total reformulação do projeto. Desta forma, a rede de comunicação desenvolvida deve ser baseada em componentes que tenham possíveis substitutos comercialmente disponíveis, de simples acesso e baixo custo.

(d) **Simples de Implementar:** Visto que o presente trabalho tem como um dos principais objetivos a redução do tempo e esforço gastos na implementação de sistemas de sensoriamento e controle, a simplificação da utilização do sistema de comunicação é um ponto crucial para atingir tal objetivo. Além disso, de modo a aumentar a chance de aceitação do mesmo, qual-

quer complicação em relação ao seu manuseio deve ser evitada. Deste modo, deve-se evitar a necessidade de realizar configurações iniciais de caráter manual para possibilitar a sua utilização, tornando a sua implementação simples e intuitiva.

(e) **Cabeamento Reduzido:** Segundo (DIFTLER et al., 2011) existe uma correlação direta entre a quantidade total de cabos existentes em um sistema e a sua confiabilidade. Desta forma, visto que cabos em um sistema multiarticulado são potenciais focos de erro, é de suma importância minimizar a quantidade dos mesmos. Além disso, a redução do cabeamento como um todo de um sistema de comunicação simplifica a sua implementação.

(f) **Alta Taxa de Transmissão:** Devido ao fato de que a rede desenvolvida será parte de um sistema de controle sem um fim especifico, o dimensionamento da taxa de transmissão da rede deve ser realizado de uma forma conservadora. Exoesqueletos robóticos muitas vezes fazem uso de sensores para adquirir sinais biológicos como EEG e EMG. Os sinais provenientes de tais sensores podem muitas vezes chegar à frequências de algumas centenas de Hz. Além disso, estes devem ser amostrados com uma precisão de pelo menos 10 ou 12 bits. Considerando uma frequência de 3 Khz de ciclo de controle, 16 sensores e um tempo de aquisição dos dados de 20% do período da malha de controle, teremos pelo menos uma necessidade de 2,88 Mbps de taxa de transmissão. Considerando a existência de outros sensores, latência entre mensagens e *overhead* pode-se considerar um valor mínimo de taxa de transmissão de 25 Mbps.

Para finalizar, de modo a contribuir para o processo de implementação de sistemas similares em projetos de outros grupos, a rede aqui desenvolvida terá seus arquivos fonte disponibilizados sob a licença GNU LGPL (FSF, 2007). O autor espera que esta iniciativa sirva como um incentivo para uma

padronização dos sistemas de sensoriamento e controle na área da robótica. Mais informações se encontram no Capítulo 5 deste documento.

1.4.2 Tecnologias

1.4.2.1 FPGA

FPGAs, ou *Field Programmable Gate Arrays*, são dispositivos semicondutores compostos por milhares de células lógicas idênticas individualmente reconfiguráveis (BARR, 1999). A arquitetura de tal célula lógica não é fixa e cada fabricante adota diferentes variações em seus produtos. Elas basicamente são compostas por blocos de memória RAM, registradores de deslocamento, flip-flops, multiplexadores e LUTs. Um exemplo de uma célula lógica (extraído de (ALTERA, 2009)) pode ser visto na Figura 2.

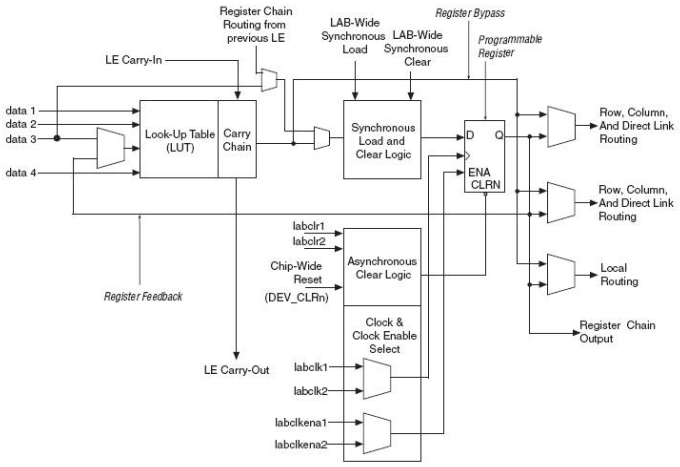


Figura 2: Célula lógica do FPGA Cyclone®IV

Estas células, por sua vez, estão interconectadas através de um conjunto chaves reconfiguráveis chamadas de *switch box*. Uma ilustração (extraída de (BETZ, s.d)) da disposição de tais chaves pode ser vista na Figura 3.

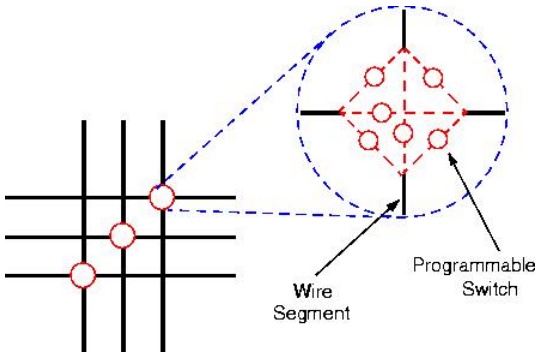


Figura 3: *Switch Box*

Desta forma, devido a estas duas características, em conjunto com a flexibilidade de configuração de suas portas de entrada e saída, é possível implementar em um FPGA basicamente qualquer circuito lógico digital (BARR, 1999). Esta capacidade o faz ser amplamente utilizado na indústria para auxiliar com o projeto de ASICs e ASSPs. Além disso, muitas vezes acaba não sendo conveniente, no ponto de vista econômico, criar uma produção de um ASIC e desta forma FPGAs são comumente utilizados diretamente em produtos finais (WAIN et al., 2006). Como este trabalho visa projetar e desenvolver um sistema com requisitos bem específicos, o FPGA foi utilizado de forma a suprir a demanda de possíveis dispositivos eletrônicos que tenham características não encontradas em produtos comercialmente disponíveis. É importante ressaltar que neste texto, a expressão *IP Core* é referente ao código utilizado para configurar o FPGA com uma função predeterminada.

1.4.2.2 Microcontroladores

Microcontroladores (μC) são circuitos integrados que contém, no mesmo encapsulamento, um processador e diversos periféricos, como memória RAM, memória Flash, conversores Analógico para Digital, controladores de PWM e temporizadores junto com diversas interfaces disponíveis, como UART, SPI, I2C entre outras (HEATH, 2003). Além disso, muitos deles têm a capacidade

de trabalhar com interrupções, que são basicamente rotinas processadas assincronamente com a rotina principal, podendo ser inicializada de acordo com algum evento predeterminado. Um exemplo de arquitetura interna, do microcontrolador PIC32MX da microchip, pode ser visto na Figura 4 (extraída de (MICROCHIP, 2010)).

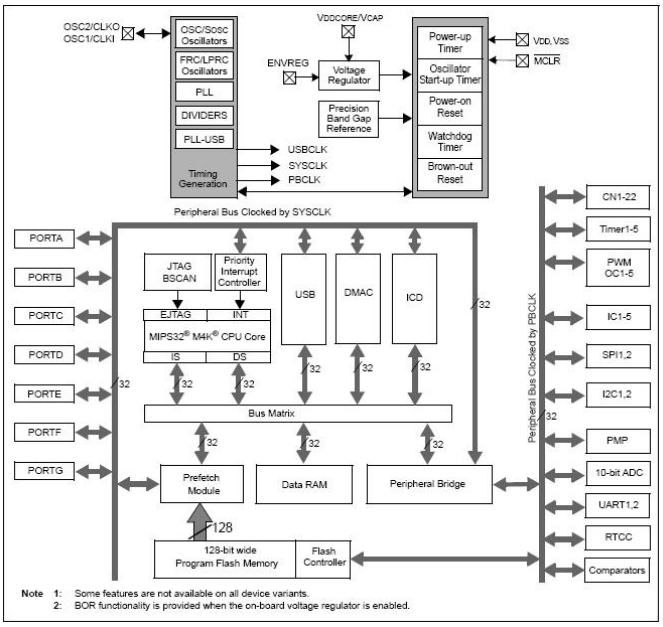


Figura 4: Arquitetura do PIC32MX

Tais dispositivos normalmente têm uma capacidade de processamento bem limitada (alguns Mhz), pois são projetados de forma a terem um custo reduzido e um baixo consumo. Pelo fato de serem normalmente utilizados para aplicações que requeiram um controle rigoroso, suas rotinas normalmente são programadas de modo a serem executadas diretamente fazendo uso das instruções nativas do microcontrolador, ou seja, sem nenhum sistema operacional. Tal prática faz com que os fabricantes (ou outras empresas) criem compiladores (normalmente fazendo uso da linguagem C de programação) que utilizem tais instruções de uma forma eficiente.

Existem também dispositivos que, além de todas essas características descritas anteriormente, possuem periféricos específicos para processamento

de sinais, como MAC e FPU, que são unidades implementadas em hardware capazes de realizar operações matemáticas utilizando poucas instruções do processador. Estes dispositivos são chamadas de DSP (*Digital Signal Processor*), ou muitas vezes de DSC (*Digital Signal Controller*), sendo que a diferença entre ambos é extremamente subjetiva, normalmente baseada na capacidade de processamento. É importante ressaltar que existem DSPs com capacidade de processamento relativamente alta, porém tais dispositivos são equivalentes a processadores de alto desempenho com instruções específicas para processamento digital. Tais dispositivos não serão incluídos nessa análise visto que não compartilham as características de flexibilidade e baixo custo dos microcontroladores convencionais, além de serem muito mais complexos de se utilizar.

No mercado existem diversos fabricantes de microcontroladores e DSPs, todos com uma grande gama de dispositivos, cada um com um conjunto de periféricos, interfaces, capacidade de processamento e arquitetura diferentes. Devido ao seu poder de processamento relativamente limitado, eles não têm capacidade de controlar sistemas complexos como os robôs descritos nesse trabalho. Porém, esta imensa diversidade de dispositivos disponíveis, junto com o baixo preço e facilidade de uso, os torna uma ferramenta extremamente poderosa para serem utilizados em sistemas embarcados de forma a criar uma interface com sensores e atuadores, além de servirem como processadores locais. Visto esta característica e o fato de que este trabalho visa projetar e desenvolver um sistema flexível de forma a poder ser integrado com os mais variados sensores e atuadores, é evidente que microcontroladores e DSPs podem ser utilizados de modo a auxiliar a suprir tal requisito. De forma a simplificar a nomenclatura utilizada, tanto DSPs como DSCs serão aqui chamados de microcontroladores.

1.5 Organização do Texto

Este trabalho está organizado da seguinte maneira: no Capítulo 2 é apresentado o projeto conceitual do sistema de sensoriamento e controle desenvolvido descrevendo os seus módulos, além de serem apresentadas as diretrizes para a sua implementação. No mesmo capítulo ainda é apresentada uma descrição da rede de comunicação R-Bone desenvolvida. No Capítulo 3 é descrita a implementação do sistema projetado, detalhando todos os aspectos técnicos relevantes de seu funcionamento, incluindo as informações pertinentes para a sua replicação e manuseio. No Capítulo 4 são apresentados os testes realizados com o sistema implementado em conjunto com uma discussão dos resultados obtidos. Para finalizar no Capítulo 5 é apresentada a conclusão deste trabalho em conjunto com um breve comentário sobre os trabalhos futuros. Adicionalmente neste último capítulo também se encontram as informações para encontrar os arquivos fonte deste trabalho disponibilizados.

2 PROJETO DO SISTEMA

Neste capítulo é descrito o projeto conceitual do sistema desenvolvido. Aqui são apresentadas as definições da arquitetura do sistema, de como o sistema deve ser organizado, e qual a sua lógica de funcionamento. O primeiro ponto aqui descrito é relativo à rede de comunicação projetada, a qual teve um impacto significativo no "formato"do sistema. Em seguida são apresentados detalhes sobre a arquitetura do sistema como um todo.

2.1 Projeto da Rede de Comunicação

Inicialmente foram avaliados alguns sistemas de comunicação existentes implementados em outros projetos. Estes foram analisados de acordo com os requisitos definidos e nenhum dentre eles se provou suficiente para cobrir tais demandas. Desta forma, uma rede de comunicação específica para sistemas robóticos complexos foi desenvolvida. A seguir será dado um detalhamento das redes de comunicação existentes consideradas e o por que de sua não-utilização. Em seguida será apresentado o projeto detalhado da rede desenvolvida.

2.1.1 Sistemas de Comunicação Existentes

Como ja apresentado no Capítulo 1, existe uma grande diversidade de sistemas de comunicação implementados em exoesqueletos e robôs humanoides. Dentre eles foram selecionados o CAN, EtherCAT, ExoNet e IEEE 1394. Para cada um desses padrões serão apresentados os motivos pela sua seleção, uma breve descrição das suas características e finalizando o motivo de sua não-utilização. Para todos os padrões foi considerado par de fio de cobre trançado como meio físico de transmissão. Uma tabela com uma análise qualitativa em relação aos requisitos pode ser visto na Tabela 1. É importante ressaltar que o requisito simplicidade de implementação não foi avaliado por ser extremamente subjetivo.

Tabela 1: Avaliação dos padrões de comunicação

Sistema	Taxa de Transmissão	Baixa Latência	Replicabilidade	Determinismo	Cabeamento Reduzido
CAN		✓	✓	✓	✓
EtherCAT	✓	✓		✓	✓
ExoNet	✓	✓		✓	✓
IEEE 1394	✓		✓	✓	✓

CAN

CAN foi selecionado para essa avaliação por ser um dos padrões mais empregados em projetos de robôs com controle distribuído. Ele é um padrão de comunicação que especifica tanto a camada de enlace quanto o meio físico. Ele trabalha com uma topologia em barramento e não exige a necessidade de um mestre ou dispositivo que controle o fluxo de comunicação da rede. Isto é possível devido à duas especificações existentes no padrão. A primeira diz que em um barramento CAN, um nível lógico 0 deve ter predominância sobre um nível lógico 1. A segunda especificação diz que todos os nós da rede devem constantemente monitorar os dados transmitidos e caso seja en-

viado um bit com nível lógico 1 e a leitura da rede acuse nível lógico 0, o Nó em questão deve parar de enviar a mensagem e esperar até que a rede esteja livre. Desta forma, como todas as mensagens devem conter um número identificador de 11 bits, se dois ou mais Nós tentarem enviar uma mensagem simultaneamente, a que possuir o identificador com o menor valor terá preferência (BOSCH, 1991). Este mecanismo é extremamente eficiente, já que cria uma arbitragem automática dos dados na rede sem haver colisão destrutiva, o que permite um melhor aproveitamento da banda disponível, já que os Nós podem enviar dados pela rede sem haver a necessidade de uma mensagem requisitando tais dados. Desta maneira, caso um Nó no sistema esteja amostrando um certo sensor, ele pode enviar valores apenas quando houver mudanças no valor lido, sem haver a necessidade de o Nó interessado neste dado ficar constantemente requisitando dados.

Outra grande vantagem deste padrão é que devido ao fato de ele ser muito bem difundido em diversas áreas de aplicação, existe uma grande variedade de circuitos integrados com a função de um controlador CAN (que inclui todo o hardware necessário para a sua implementação) disponíveis no mercado por diversos fabricantes. Estes circuitos integrados têm um custo relativamente baixo e por terem uma alta disponibilidade, há uma simplificação no processo de replicação e manutenção do sistema.

Sumarizando, as principais características de um padrão de comunicação CAN são: máxima taxa de transferência de dados de 1 Mbps (com um barramento de 40m) e *overhead* de 44 bits para uma quantidade máxima de 64 bits de dado (para uma mensagem com um formato não estendido). Não existe um número definido de Nós possíveis na rede, sendo este número definido de acordo com o transceiver utilizado (BOSCH, 1991).

O CAN é um padrão extremamente robusto e que faz um ótimo uso de

sua banda de transmissão disponível, dispensando a necessidade de que o mestre arbitre as informações enviadas pelos nós. Porém, a sua máxima taxa de transmissão de 1 Mbps é considerada demasiadamente baixa para o sistema desenvolvido já que isto pode criar limitações para aplicações que requeiram uma grande quantidade de dados transmitidos. Múltiplos canais paralelos poderiam ser utilizados, porém isso demandaria uma quantidade demasiada de cabos e limitaria a escalabilidade do sistema. Para finalizar, a sua implementação seria dificultada devido às múltiplas interfaces de rede utilizadas na visão do Processador Central.

EtherCAT

Dentre as diferentes variações de Ethernet *Real-Time* existentes, EtherCAT foi considerada a opção com características mais interessantes para o presente trabalho. Além disso, alguns projetos como ROTTO e Justin já obtiveram sucesso com sua implementação. EtherCAT é um padrão de comunicação similar à Ethernet, porém com algumas modificações. Ao contrário da Ethernet, EtherCAT permite que uma única mensagem contenha dados endereçados a diversos Nós em uma rede. Deste modo, há uma melhor utilização da banda disponível, já que em muitos sistemas as mensagens transmitidas têm uma quantidade de dados muito pequena se comparada ao tamanho mínimo da mensagem de Ethernet. Este padrão pode ser implementado em uma rede com topologias tanto em Linha quanto Anel, fazendo uso de um processamento *on the fly* (ETG, 2009). Atualmente a única taxa de transmissão suportada é 100 Mbps (PRYTZ, 2008) e, se for utilizado como meio físico par de fios de cobre trançados, a máxima distância entre dois Nós é de 100m (ETG, s.d.).

Atualmente existem apenas duas empresas que fabricam ASICs para a

sua implementação. A primeira se trata da Beckhoff, que disponibiliza comercialmente dois controladores EtherCAT, o ET1100 e o ET1200. A segunda é a Hilscher que fabrica um controlador de rede multiprotocolo denominado netX que pode ser utilizado para diversos padrões de comunicação. Neste controlador também está incluído um processador ARM9 com a capacidade máxima de processamento de 200Mhz (ETG, 2009). Ambos os ASICs não são encontrados disponíveis em distribuidores convencionais de componentes eletrônicos como DigiKey, Farnell entre outros, apenas sendo possível adquiri-los diretamente com o fabricante ou com poucos distribuidores. Para finalizar também existe a opção de utilizar um *IP Core* para FPGAs desenvolvido pela Beckoff, porém este é apenas disponibilizado para produções de larga escala (ETG, 2009).

EtherCAT tem características interessantes, pois a mesma faz um melhor uso dos pacotes da Ethernet permitindo que todos os Nós possam ser endereçados em uma única mensagem, além de não ter a necessidade de o uso de um Switch, Roteador ou qualquer outro sistema gerenciador de mensagens. Adicionalmente tem uma taxa de transmissão de 100 Mbps, o que satisfaz o requisito proposto. Porém a sua implementação é restrita a ASICs fornecidos por poucos fabricantes, ou IP Cores para FPGA de alto custo e difícil acesso. Para finalizar, o autor desse trabalho não conseguiu ter acesso a nenhum ASIC disponível no mercado, já que os mesmos não podem ser comprados em distribuidores convencionais de componentes eletrônicos, impossibilitando assim a sua implementação. Ele entrou em contato um representante da Beckhoff no Brasil, porém não houve sucesso na tentativa de compra de componentes e *kits* de desenvolvimento.

ExoNet

ExoNet não tem uma especificação muito bem definida e nem é amplamente utilizado em sistemas robóticos, porém foi analisado por ter sido projetado especificamente para aplicações similares a este trabalho. ExoNet é um padrão desenvolvido na Universidade de Berkeley e implementado no exoesqueleto BLEEX. Este padrão utiliza topologia em Anel e faz uso de um processamento *on the fly* para otimizar o uso da banda de comunicação. Ele é baseado em um único circuito integrado denominado CY7C924ADX Hotlink, fabricado pela Cypress. A sua taxa máxima de comunicação é de 200Mbps e uma máxima distância não especificada, dependendo apenas do seu meio físico utilizado. A máxima quantidade de Nós em uma rede se limita a 8 devido ao protocolo de comunicação utilizado, porém este pode ser facilmente superado com uma simples mudança em suas definições. Assim como EtherCAT, neste padrão um único pacote pode conter mensagens para diversos nós, porém o seu formato é mais flexível, desta forma podendo ser otimizado para diferentes aplicações (KIM; KAZEROONI, 2004).

ExoNet tem características muito parecidas com EtherCAT, porém tem uma maior flexibilidade no formato dos pacotes e um maior potencial em relação à taxa de transmissão. Porém, é baseada em um único CI (Cypress CY7C924ADX) que tem um custo unitário de aproximadamente 100 dólares e não é facilmente substituível por outras formas como FPGAs ou microcontroladores. Caso este CI seja descontinuado pelo fabricante, o sistema ficará comprometido e terá que ser reprojetado, inviabilizando todos os Nós já construídos.

IEEE 1394

Alguns trabalhos recentes obtiveram sucesso na implementação de IEEE 1394 em um controle *real time* em sistemas robóticos. Além disso, este padrão consegue chegar a altas taxas de transmissão e tem uma grande flexibilidade na topologia empregada.

Existem diversas versões da especificação do IEEE 1394 e dependendo desta, a taxa de transmissão pode variar de 100Mbps até 3.2Gbps (SARKER et al., 2006; THIENPHRAPA; KAZANZIDES, 2011). Existem dois modos de transmissão de dados, o isochronous e assíncrono. No primeiro existe uma garantia de banda transmissão porém não há checagem na integridade dos dados. Este modo é comumente utilizado para transmissão de dados que tenham requisitos de temporização bem rigorosos sem a necessidade da integridade de todos os bits transmitidos, como áudio e vídeo (IEEE, 1996).

No modo assíncrono, existem uma garantia na entrega dos dados corretamente, porém para tal o tempo de transmissão não é garantido. Este modo é mais apropriado para implementação de controle visto que mensagens entregues com dados errados podem levar a respostas não esperadas do sistema. Existe uma boa diversidade de componentes de vários fabricantes para realizar a sua implementação. Um dos grandes problemas associados à sua utilização está no fato de que as mensagens são enviadas separadamente e não podem endereçar mais de um Nó por vez. Apesar da alta taxa de transmissão do IEEE 1394, a utilização de diverso Nós se torna inviável, visto que um dos grandes focos de latência se encontra na conexão do Processador Central com a rede como demonstrado em (THIENPHRAPA; KAZANZIDES, 2011).

2.1.2 Rede Desenvolvida

Aqui será apresentada uma descrição das características da rede projetada, assim como os aspectos técnicos relevantes que influenciaram no desenvolvimento da mesma. Esta, por sua vez, foi denominada R-Bone (*Robotic Backbone*). É importante ressaltar que apesar de aqui ser descrita a camada de enlace da rede, existe uma sobreposição da mesma com a camada de aplicação, como será comentado.

2.1.2.1 Topologia de Rede

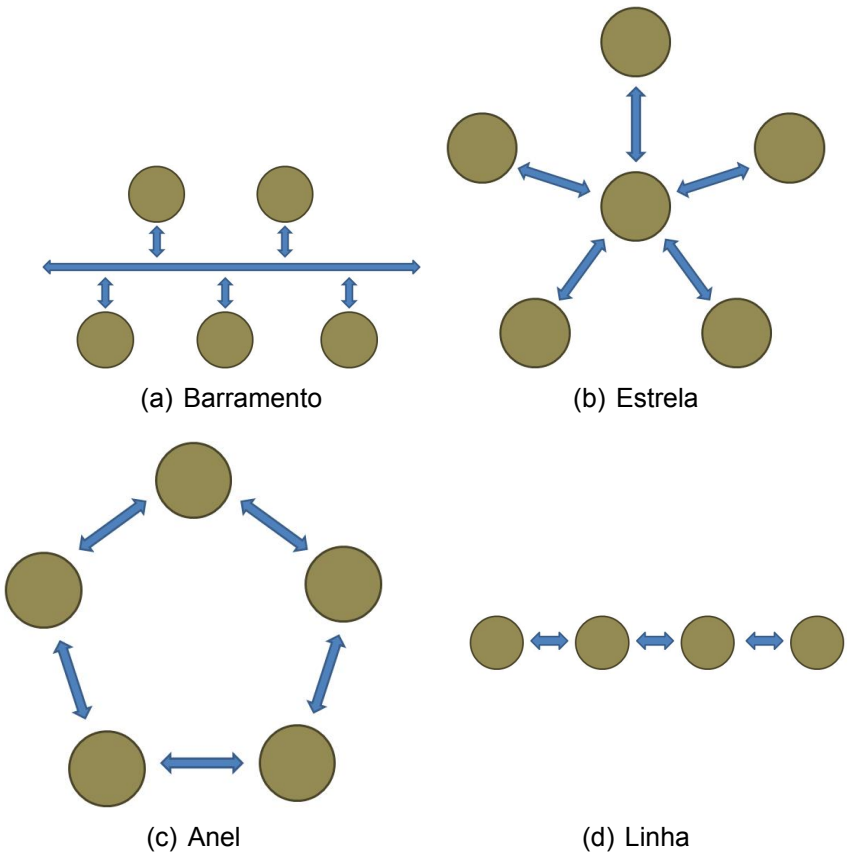


Figura 5: Topologias de rede

Um dos principais pontos que definem o funcionamento e características de uma rede de comunicação é a sua topologia. Esta basicamente se define

como a forma em que todos os sistemas da rede estão interconectados fisicamente ou logicamente (RATHA, s.d). No presente caso será feita uma análise em relação à topologia física da rede. Na Figura 5 são ilustradas algumas topologias cujas características serão descritas e posteriormente comparadas.

É importante ressaltar que existem mais topologias além das aqui descritas, porém, por terem visivelmente uma quantidade exagerada de conexões (visto que o presente trabalho visa reduzir ao máximo a quantidade de cabos no sistema) ou por serem consideradas uma mistura das topologias apresentadas, não foram avaliadas. Nesta discussão todos os dispositivos em uma rede serão denominados de nós, o que não faz uma correlação exata com os Nós apresentados anteriormente, já que o Processador Central se excluí deste grupo.

Barramento

Em uma topologia de Barramento, todos os nós estão conectados através de um único meio físico e contínuo. Em grande parte das aplicações tal meio físico consiste de um par de fios trançados com sinal diferencial, ou um cabo coaxial. As principais vantagens desta topologia são a reduzida quantidade de cabos que cruza o sistema e fato de que os dados enviados de um nó para outro cruzam a rede de comunicação sem nenhum intermediário, o que faz o sistema ter uma latência reduzida e bem controlada. Em contrapartida, normalmente padrões de comunicação que utilizem a topologia em Barramento têm uma taxa de transmissão máxima inversamente proporcional à quantidade de nós presentes na rede e ao comprimento do barramento (SULLIVAN, 2004).

Estrela

Nesta topologia todos os nós estão interconectados através de conexões ponto-a-ponto a um dispositivo único. Este, por sua vez, faz o gerenciamento das mensagens da rede podendo ele acontecer de dois modos: no primeiro as mensagens são repassadas apenas para o nó destinatário da mesmo, enquanto no segundo ela é repassada para todos os nós presentes e estes são encarregados de diferenciar quais mensagens são endereçadas a eles. Esta topologia permite uma grande quantidade de nós em uma rede sem comprometer a sua capacidade de transmissão máxima já que fisicamente só existem conexões ponto-a-ponto. Em contrapartida, dependendo da disposição dos nós, esta topologia requer uma quantidade muito maior de cabos em relação às outras existentes (SULLIVAN, 2004). Um exemplo seria um sistema onde todos os nós estão dispostos em uma linha não circular. Neste caso, mesmo que o dispositivo gerenciador de mensagens esteja localizado em uma posição central, diversos cabos irão percorrer o sistema em paralelo, ao contrário do que aconteceria em uma topologia em barramento, onde haveria apenas um cabo percorrendo todos os pontos do sistema. Para finalizar, este dispositivo gerenciador de mensagens sempre inclui uma mínima latência na transmissão de dados.

Linha

Como pode ser visto na Figura 5(d), em uma topologia de Linha os dispositivos estão interconectados por conexões ponto-a-ponto, de uma forma sequencial, de acordo com a disposição física. Nesta topologia, assim como na Estrela, não há redução na capacidade de transmissão máxima da rede em relação à quantidade de nós existentes já que só existem conexões pon-

to-a-ponto. Em contrapartida, cada nó que uma mensagem percorre adiciona latência na transmissão. Desta forma, a latência máxima da rede está diretamente relacionada com a quantidade de nós presentes (FJELDBO, 2005). Para reduzir este problema, muitas dos sistemas de comunicação fazem uso de um processamento da informação *on the fly*, onde os nós repassam as mensagens adiante antes de receber o fim das mesmas, modificando-as dinamicamente. Outra característica negativa é o fato de que se um nó falha, o sistema ficará "dividido", comprometendo seu correto funcionamento, o que pode ser um fator crítico em sistemas que requerem alta confiabilidade (FJELDBO, 2005).

Anel

Topologia em Anel nada mais é que uma topologia em Linha com uma conexão extra entre os nós posicionados nas extremidades. Ela tem as mesmas características que a topologia em Linha descrita anteriormente, porém esta conexão extra pode trazer dois benefícios para a rede. O primeiro é a criação de uma redundância na comunicação (SULLIVAN, 2004). Outro possível benefício é um melhor uso da banda de transmissão, utilizando o processamento *on the fly* para um mesmo nó enviar e receber dados na mesma mensagem, sem ela ter que percorrer duas vezes todos os nós.

Comparação

Este trabalho visa desenvolver um sistema de controle que será utilizado em dispositivos munidos de membros, como robôs humanoides, ou exoesqueletos. Desta maneira, o Processador Central ficará localizado em uma posição centralizada em relação aos membros. Os Nós estarão distribuídos pelo sistema, porém majoritariamente concentrados nos membros e dispostos

em linha. Assim sendo, fica evidente que a topologia em estrela não é uma boa escolha pois a mesma demandaria uma grande quantidade de cabos cruzando os membros em paralelo. A topologia em Anel, por sua vez, pode ser implementada em um sistema com nós alinhados com apenas dois pares de fios presentes em cada ponto, um a mais que em uma topologia em Linha decorrente da conexão extra. Esta desvantagem é aceitável, visto o ganho de eficiência e robusteza. Além disso, ambos os cabos podem ser reduzidos à apenas um se considerarmos a possibilidade de diversos fios (ou pares de fios no caso de sinais diferenciais) dentro de um mesmo cabo (e.g CAT5). Para finalizar, a topologia em Barramento é atraente para esta aplicação já que a mesma tem uma quantidade de cabos extremamente reduzida, uma latência bem controlada e as aplicações em questão não terão uma grande quantidade de nós em cada membro.

Desta forma, ficam as topologias e Anel e Barramento como melhores escolhas para a rede em questão. Como comentado anteriormente, topologias em estrela fazem uso de comunicações ponto-a-ponto, podendo alcançar uma maior distância e taxa de transmissão do que padrões que façam uso de topologia em barramento. Em contrapartida, possuem uma latência variável de acordo com a quantidade de nós presentes, reduzindo assim a performance e dificultando uma escalabilidade. Visto que grande parte dos sistemas robóticos possuem um tamanho limitado a poucos metros de raio e para pequenas distancias a topologia em barramentos pode alcançar taxas de transmissão aceitáveis, ela foi escolhida para ser a topologia da rede desenvolvida.

De forma a reduzir a quantidade de cabos e fazer o uso de comunicações em paralelo, será implementado um sistema multicanal, sendo visado o uso de um canal para cada membro, porém não obrigatoriamente. É importante ressaltar que tal implementação não pode ser denominada de uma topologia em

Árvore (que se trata de uma mistura das topologias Barramento e Estrela), já que a princípio os canais serão totalmente separados, tendo apenas o Processador Central como nó em comum. A Figura 6 ilustra a configuração proposta para um sistema com quatro canais.

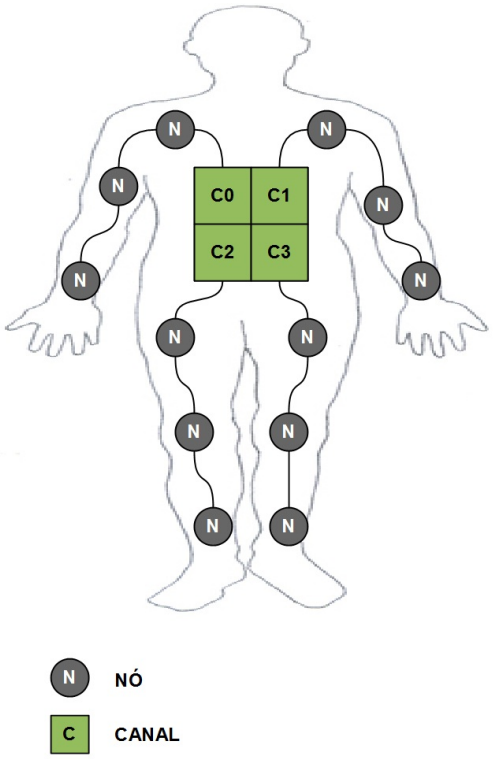


Figura 6: Topologia da rede desenvolvida

2.1.2.2 Camada Física

Foram considerados dois padrões para implementar a topologia em barramento da rede projetada: RS-485 e Multipoint LVDS (M-LVDS). O primeiro é vastamente empregado em aplicações industriais e existe uma grande quantidade de transceivers disponíveis no mercado de diversos fabricantes. Tradicionalmente foi projetado para comportar um máximo de 32 nós em seu barramento, porém os fabricantes de circuito integrado modificaram a carga adicionada pelos tranceivers de modo a poder superar tal número. No atual

estado da arte, o transceiver com maior capacidade de transmissão é da Maxim, com uma taxa máxima de 40 Mbps (MAXIM, 2010). Não existe uma definição da distância máxima que se pode alcançar com um barramento RS-485, porém o mesmo pode chegar a até 1 Km para baixas taxas de transmissão.

M-LVDS tem um conjunto de aplicações mais restrito, assim como um menor número de transceivers disponíveis. Tem uma capacidade de suportar até 32 nós em um mesmo barramento e uma taxa de transmissão máxima de 500 Mbps. Não existe uma definição para a distância de transmissão, porém esta normalmente é menor do que alcançada pelo RS-485 para uma mesma taxa de transmissão. Isto é decorrente do fato de que o nível de tensão utilizado pelo M-LVDS é menor, sendo mais penalizado pelas perdas decorrentes da transmissão (TI, 2003a).

Dado que nas aplicações objetivadas pelo sistema desenvolvido não haverá a necessidade de distâncias de transmissão maiores que 3 metros, o M-LVDS se faz mais atrativo pela sua máxima taxa de transmissão de 500Mbs. Desta forma o M-LVDS tem características ideais para esta aplicação como alta taxa de transmissão, baixo custo, capacidade de multi-ponto e alta imunidade eletromagnética. A taxa de transmissão escolhida para ser utilizada na rede é de 50 Mbps.

2.1.2.3 Camada de Enlace

O primeiro ponto definido da camada de enlace foi o formato de transmissão das mensagens (também referidas aqui como pacotes). Estas devem ser compostas por um cabeçalho, por um campo de dados variável e por 16 bits de *Cyclic Redundancy Check* (CRC). Este último é utilizado para conferir a integridade das informações recebidas e para tal deve ser utilizado o seguinte

polinômio.

$$x^{16} + x^{15} + x^2 + 1 \tag{2.1}$$

O cabeçalho deve ser composto por quatro informações diferentes, todas com o tamanho de um byte: endereço, comando, identificação (ID) e comprimento do dado. O endereço, como o próprio nome diz, deve conter o endereço do destinatário da mensagem porém apenas os 5 bits menos significativos devem ser utilizados para tal. Os três bits mais significativos devem ser utilizados para indicar o canal cujo destinatário da mensagem em questão se encontra. O comando, por sua vez, deve conter a informação do tipo da mensagem enviada. Esta informação deve ser definida na camada de aplicação, porém existem quatro comandos reservados que não podem ser utilizados. O primeiro é o *acknowledgement* (ACK), o qual é utilizado como resposta a todas as mensagens que tenham sido recebidas com um CRC válido. De uma forma complementar, o segundo comando restrito é o *negative acknowledgement* (NACK), o qual deve ser enviado como resposta para todas as mensagens que tenham sido recebidas com um CRC inválido. Os dois comandos remanescentes são o CONFIG e o POLL cujas funções serão descritas mais adiante. De forma a aumentar a eficiência da rede, as mensagens cujos comandos sejam ACK, NACK e POLL devem ser compostas exclusivamente pelos campos de endereço e comando. Consequentemente, elas não precisam receber um ACK ou NACK de resposta como forma de confirmação ao seu conteúdo.

O campo de ID deve carregar um identificador da mensagem. Dentre as suas funções a principal é servir como uma forma de o Processador Centrar identificar quais mensagens recebidas por ele são repostas a suas solicitações enviados aos Nós. Para finalizar, o comprimento do dado deve conter

a quantidade de bytes que compõe o campo de dados. Este último, por sua vez, tem um tamanho variável de zero a 255 bytes e deve carregar informações complementares ao comando da mensagem. A figura 7 ilustra o formato do pacote descrito.

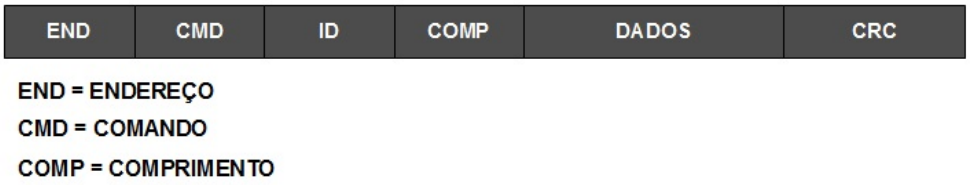


Figura 7: Formato do pacote de transmissão da rede

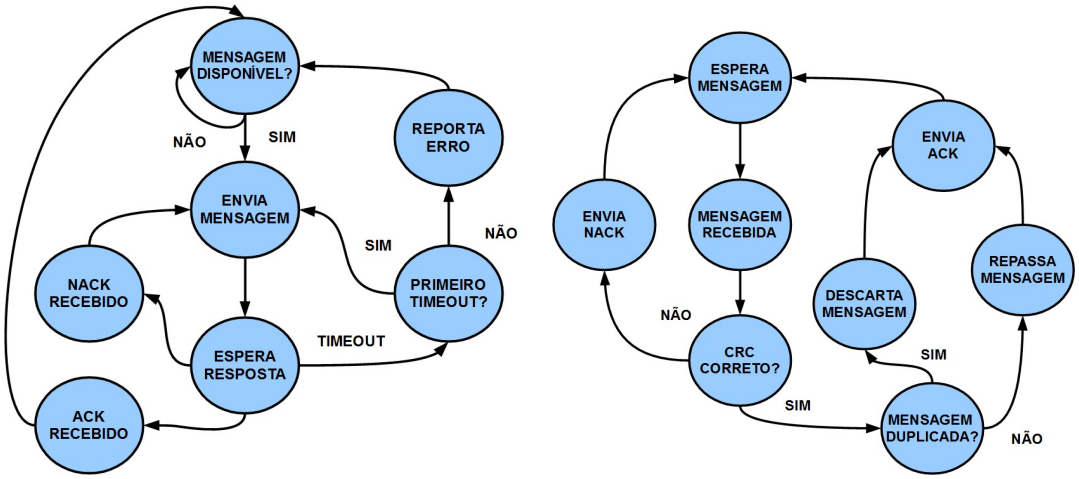
Após definido o formato dos pacotes transmitidos, deve-se definir o formato de transmissão de dados. Estes devem ser transmitidos um byte por vez, com o bit menos significativo (LSB) sendo enviado primeiro. Cada byte deve ser precedido por um START bit o qual deve sempre estar em nível lógico 1. Complementarmente deve haver um STOP bit ao final de cada byte, o qual sempre deve estar em nível lógico 0. Estes são utilizados para garantir a existência de uma transição de nível lógico entre cada byte de modo ao receptor poder efetuar a recuperação dos dados. Para finalizar deve haver um bit de RESET entre o bit mais significativo (MSB) do byte transmitido e o STOP bit. Este, por sua vez, deve estar em nível lógico 1 sempre que for transmitido um byte de endereço. Apesar de este mecanismo ser comumente utilizado na transmissão de dados entre microcontroladores de forma a reduzir o *overhead* de processamento, aqui ele tem um objetivo diferente. O RESET bit foi incluído unicamente para garantir que a interface de rede de todos os Nós esteja em um estado conhecido sempre que a transmissão de uma nova mensagem for inicializada. A Figura 8 ilustra o formato descrito.

Como já comentado no Capítulo 1, um dos pontos críticos considerados no desenvolvimento da rede de comunicação é manter um nível de latência

Figura 8: Formato de transmissão dos dados

reduzido. De modo a tal, foi definido que cada canal de rede deve conter duas linhas de comunicação próprias independentes. Ambas seguem o conceito de mestre-escravo para realizar o controle de fluxo e arbitragem onde existe apenas um mestre na rede e diversos escravos. Todas as trocas de dados são sempre inicializadas pelo mestre independente do real sentido da troca de dados.

A primeira linha, denominada de Linha Bus, é utilizada para distribuir informações enviadas pelo Processador Central para os Nós. O mestre da linha fica no modo de espera até que uma mensagem precise ser enviada. Quando esta se encontra disponível, o mestre envia a mensagem e o escravo endereçado deve responder com um ACK caso o CRC recebido seja igual ao calculado. Em caso contrário um NACK deve ser enviado como resposta e o mestre reenvia a mensagem até que esta seja entregue com sucesso. Os escravos sempre devem guardar o cabeçalho da última mensagem recebida com sucesso e compará-lo com as recebidas posteriormente. Caso ambas sejam idênticas, ele envia um ACK como resposta, porém descarta a mensagem. Para finalizar, o mestre da Linha Bus tem um temporizador de um microssegundo para controlar o tempo decorrido entre o envio de uma mensagem e a sua resposta (ACK ou NACK). Caso o contador expire, um sinal de *timeout* é gerado e a mensagem considerada perdida. O mestre então envia novamente a mensagem. Se o problema persistir, a mensagem é descartada e um erro é reportado ao Processador Central. Na Figura 9 pode ser vista uma ilustração da lógica descrita.



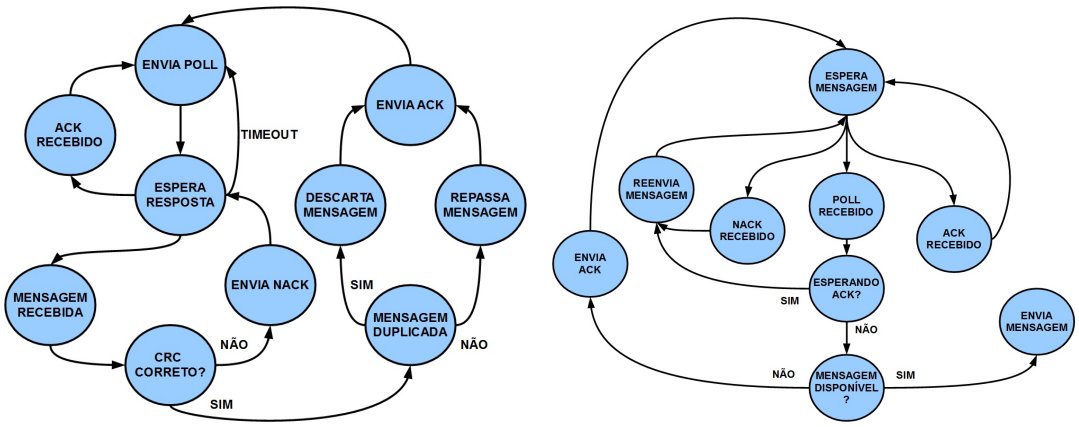
(a) Mestre (b) Escravo

Figura 9: Lógica de funcionamento da Linha Bus

A segunda linha, denominada Linha Poll, tem uma lógica de funcionamento um pouco mais complexa. A sua principal função é coletar os pacotes disponíveis nos Nós e repassá-los para o Processador Central. De forma que este processo seja feito de uma maneira ordenada, o mestre da linha inicialmente envia uma mensagem com um comando CONFIG para todos os 32 possíveis endereços existentes no canal. Estes, por sua vez, devem responder com um ACK sempre que receberem uma mensagem com tal comando. De uma forma similar à Linha Bus, o mestre da Linha Poll utiliza um temporizador de um microssegundo de forma a controlar o tempo de espera por uma resposta. Quando este se expira, a mensagem é considerada perdida e o respectivo endereço se torna inativo. Com este procedimento, o mestre consegue mapear o endereço de todos os Nós existentes em seu canal.

Passada a etapa de inicialização, o mestre entra em modo de operação normal onde ele ciclicamente envia uma mensagem com o comando POLL para todos os escravos mapeados. Estes devem responder um pacote de dados caso o mesmo se encontre disponível, ou com um ACK no caso contrário. Se o mestre receber um pacote de dados com um CRC válido ele repassa o mesmo para o Processador Central e envia um ACK para o escravo em

questão. No caso contrário, a mensagem é descartada e um NACK é enviado como resposta. Sempre que uma mensagem do tipo POLL é enviada, o mestre utiliza o temporizador para controlar o tempo de resposta. Se o mesmo expira ele passa para o próximo escravo na lista. O mestre armazena o cabeçalho da última mensagem válida recebida de cada escravo presente. Caso uma mensagem seja recebida com o mesmo cabeçalho armazenado, o mestre descarta a mensagem e responde com um ACK. Os escravos devem armazenar uma cópia das mensagens enviadas em um buffer e descartá-las apenas mediante a recepção de um ACK. Enquanto uma mensagem estiver armazenada, ela deve sempre ser enviada mediante a recepção de um comando POLL. Na Figura 10 pode ser vista uma ilustração da lógica descrita, excluindo-se a rotina de mapeamento inicial.



(a) Mestre (b) Escravo

Figura 10: Lógica de funcionamento da Linha Poll

Com o formato de comunicação descrito, o funcionamento da Linha Bus e da Linha Poll não depende do tempo de resposta dos Nós às solicitações enviadas, desta forma não comprometendo a performance do sistema como um todo.

2.1.3 Análise de Determinismo da Rede

Um dos requisitos da rede desenvolvida é se comportar de uma forma determinística mediante qualquer tipo de erro de comunicação. Deste modo, uma análise comportamental deve ser feita para os possíveis erros que venham a acontecer. Nesta análise será considerado que o CRC é capaz de detectar todos os erros provenientes de mensagens corrompidas. A seguir será apresentado uma lista com todos os erros possíveis identificados e uma breve descrição de como a rede irá se comportar.

Erros na Linha Bus

(I) **ACK perdido pelo mestre:** Caso um ACK seja perdido pelo mestre da linha, o temporizador irá expirar e a mensagem será reenviada novamente. Visto que a mensagem já foi recebida com sucesso pelo escravo, o mesmo terá salvo o cabeçalho da mensagem. Ao realizar a comparação, a nova mensagem será considerada duplicada e consequentemente descartada. Um novo ACK será enviado.

(II) **NACK perdido pelo mestre:** No caso da perda de um NACK pelo mestre, será gerado um sinal de *timeout* e a mensagem será reenviada. Visto que o comportamento do mestre foi idêntico ao caso de que o NACK houvesse sido recebido, a rede continua em operação normal.

(III) **Mensagem Perdida pelo escravo:** Caso uma mensagem seja perdida pelo escravo, o mesmo não irá enviar uma resposta referente à mesma. Desta forma o temporizador do mestre irá expirar e a mensagem será reenviada. Visto que a mensagem não foi recebida com sucesso anteriormente, o escravo pode tratá-la normalmente.

Erros na Linha Poll

(I) **ACK perdido pelo mestre:** Caso um ACK seja perdido pelo mestre da linha, o temporizador irá expirar e um POLL será enviado para o próximo escravo na lista. Visto que o escravo não pretendia enviar nenhuma mensagem a princípio para o mestre, a rede continua operando normalmente.

(II) **Mensagem perdida pelo Mestre:** No caso da perda de uma mensagem pelo mestre, será gerado um sinal de *timeout* e um POLL será enviado para o próximo escravo na lista. Visto que um ACK não foi enviado de resposta para o escravo referente à mensagem perdida, esta continuará armazenada em seu buffer e consequentemente será enviada no próximo ciclo.

(III) **ACK perdido pelo escravo:** Quando um ACK é perdido por um escravo, a mensagem em seu buffer não é descartada, sendo enviada novamente no próximo ciclo. Visto que o mestre enviou um ACK anteriormente, a mensagem em questão já foi recebida com sucesso e seu cabeçalho salvo. Quando a mensagem duplicada for recebida, um ACK será enviado para o escravo e a mensagem descartada pelo mestre.

(IV) **NACK perdido pelo escravo:** Caso um NACK seja perdido pelo escravo, o mesmo irá ficar no modo de espera. Visto que o mestre estará esperando um reenvio da mensagem, o temporizador irá expirar e um POLL será enviado para o próximo escravo na lista. Como o escravo não recebeu um ACK, a mensagem se encontra armazenada em seu buffer e será enviada no próximo ciclo novamente.

(V) **POLL perdido pelo escravo:** No caso de um POLL perdido por um escravo, um sinal de *timeout* será gerado no mestre e este irá enviar

um POLL para o próximo escravo na lista, continuando com a operação normalmente.

2.2 Projeto do Hardware

Para fins de organização, primeiramente será apresentada uma visão macroscópica do sistema de acordo com as especificações definidas, com blocos representando os pequenos subsistemas existentes e uma explicação de como o sistema deve se comportar como um todo. Posteriormente, cada bloco será detalhado, apresentando cada arquitetura específica selecionada para executar suas funções e as diretrizes para a sua implementação. É importante ressaltar que esta seção foi explicitamente separada da implementação real de forma a servir como um guia para a replicação deste sistema. Visto que o mesmo pode ser construído de diversas maneiras, as tecnologias utilizadas para a sua implementação não são de caráter fixo, porém algumas sugestões são dadas, as quais foram utilizadas neste trabalho.

2.2.1 Sistema Conceitual

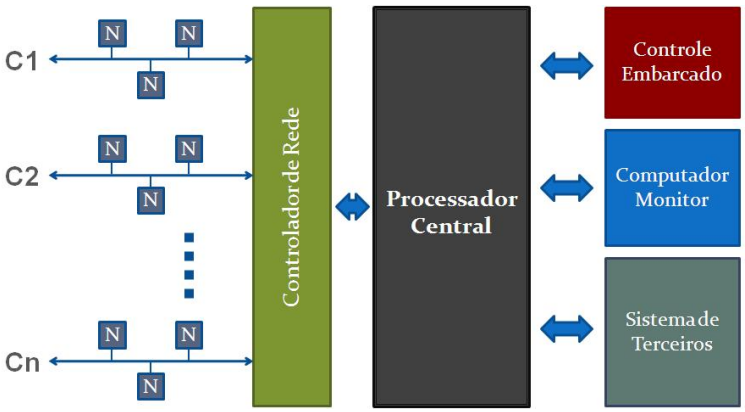


Figura 11: Arquitetura do sistema

A Figura 11 ilustra a arquitetura do sistema projetado. O bloco principal do sistema é o Processador Central. Nele deve ser processada a rotina principal de controle e é o responsável por gerenciar todo o sistema. Em uma condição normal de operação, o Processador Central deve pedir para os Nós os dados necessários para suas rotinas. Feito isto ele deve enviar ordens para os Nós, que controlarem os atuadores existentes, de forma a efetuar alguma ação determinada por sua malha de controle. Isto deve acontecer ciclicamente com um período definido.

Para realizar a comunicação, deve existir um Controlador de Rede, cuja função é gerenciar a rede projetada de um modo transparente na visão do Processador Central. De modo geral, as únicas tarefas relacionadas à comunicação que devem ficar a encargo do Processador Central são as que tenham um caráter variável de acordo com a aplicação, como a montagem das mensagens, o gerenciamento das solicitações e a interpretação das respostas. Todas as outras tarefas como gerenciamento de fluxo, controle de erros e temporização devem ser implementadas no Controlador de Rede.

Como o sistema é intencionado para ser portátil, o mesmo deve conter um Controle Embarcado que se comunicará diretamente com o Processador Central. Tal dispositivo irá ser a principal interface entre os usuários e o sistema. Com este controle o usuário será capaz de inicializar o sistema, interromper seu funcionamento e efetuar pequenas configurações. Para uma intervenção de caráter mais complexo, deve haver uma interface com um computador externo possibilitando uma supervisão do sistema. Este computador externo será aqui referido como Monitor.

Embora este sistema não esteja sendo projetado para interagir com nenhum sistema de terceiros em específico, é necessário ele tenha disponível as principais interfaces usadas comercialmente. Esta característica é de ex-

trema importância visto que existem diversos equipamentos que são inviáveis de serem adaptados para serem incluídos na rede desenvolvida.

2.2.2 Processador Central

O Processador Central deve cuidar da maior parte das rotinas de controle (algumas rotinas simples podem ser processadas nos Nós) e deste modo ele deve ter uma capacidade de processamento razoável, além de um tamanho reduzido já que o sistema deve ser portátil. Ele deve ter disponível, Ethernet, USB e Portas Seriais, já que estas são as principais interfaces utilizadas comercialmente nos tempos atuais. Para finalizar, ele deve ter disponível uma interface de comunicação com uma alta taxa de transmissão de dados para se comunicar com o Controlador de Rede, como PCI ou PCI Express. É preferível que ele seja capaz de rodar um Sistema Operacional, como o Linux, para fazer uso de todos os recursos que o mesmo provê (como diversas *threads* simultâneas) e para ser possível utilizar programas de terceiros, como o Matlab/Simulink.

Para satisfazer todos os requisitos, o formato PC/104+ foi escolhido para ser a placa do Processador Central. Basicamente se trata de um computador convencional com um formato compacto e tem disponível tanto uma interface PCI quanto uma ISA através de conectores, de forma que as placas de expansão possam ser empilhadas em paralelo e não na perpendicular como em computadores tradicionais.

No nível de software, deve ser utilizado um sistema operacional Linux com um *patch* Xenomai. Deve ser utilizado um driver para interfacear a aplicação do sistema com o Controlador de Rede implementado. De modo a facilitar a implementação de rotinas de controle é sugerível o desenvolvimento de uma biblioteca em C, que faz uso do driver, contendo funções de acesso direto aos

Nós. Uma ilustração da arquitetura interna do Processador descrita pode ser vista na Figura 12.

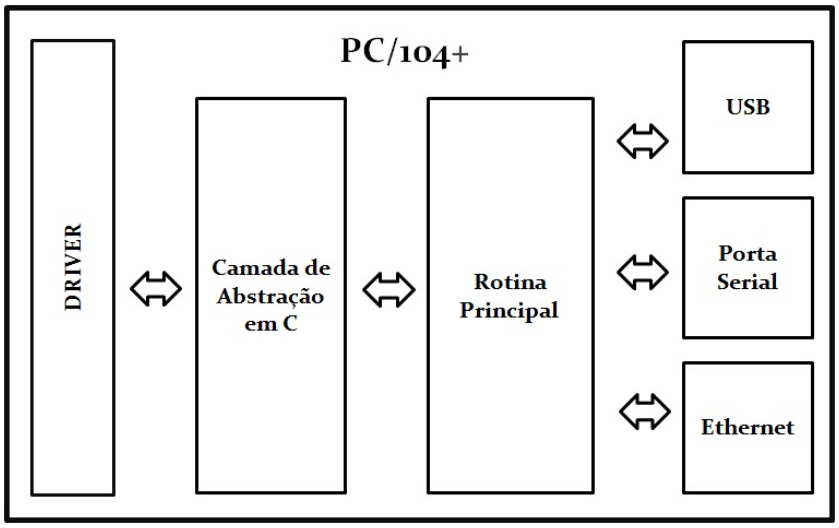


Figura 12: Arquitetura do Processador Central

2.2.3 Nós

Os Nós da rede devem servir como uma interface intermediária entre os sensores/atuadores e o Processador Central. Para tal, eles devem conter um processador local que seja capaz de lidar com rotinas de controle simples e pré-processar dados amostrados. Este, por sua vez, pode ser um microcontrolador ou algum tipo de *soft-core* implementado em um FPGA. Adicionalmente ele deve conter todo hardware necessário para lidar com os sinais dos sensores e atuadores definidos para estarem disponíveis através dele. Nele também deve estar presente o circuito que irá disponibilizar a interface com o Controlador de Rede cuja pode ser implementada utilizando um FPGA de baixo custo em conjunto com um transceiver M-LVDS. Este último é de caráter obrigatório. Na Figura 13 pode-se ver o modelo da arquitetura de um Nó.

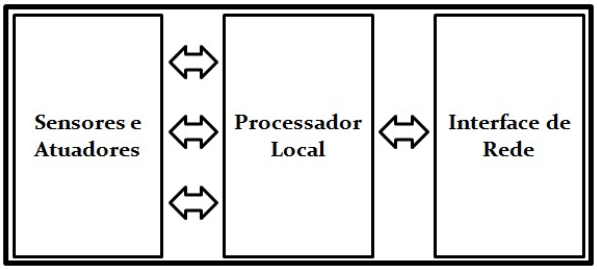


Figura 13: Arquitetura dos Nós

2.2.4 Controlador de Rede

Para a implementação do Controlador de Rede é sugerível a utilização de um FPGA, visto que não foi identificada nenhuma outra tecnologia capaz de tal. Ele deve ter uma interface PCI para se conectar ao Processador Central e pode ter um número variável de canais de rede. Para implementar a interface PCI podem ser utilizados ASICs específicos ou o próprio FPGA. No último caso, se for preciso, devem ser utilizados circuitos integrados do tipo *quickswitch* para realizar a transdução do nível de tensão utilizado na PCI (5V) com o nível de tensão disponível nos FPGAs modernos (atualmente 3,3V). Adicionalmente, é sugerível que a lógica de controle seja implementada de uma forma modular. O Controlador de Rede não deve ser implementado utilizando lógicas de controle que não tenham um tempo de resposta bem determinado como processadores de modo geral. Para implementar o nível físico da rede devem ser utilizados transceivers M-LVDS. Para finalizar o gerenciamento das Linhas Poll e Bus de todos os canais, incluindo as temporizações e cálculo de CRC devem obrigatoriamente ser implementados pelo Controlador de Rede. Adicionalmente o mesmo deve ser responsável pelo roteamento das mensagens para o canal correto de uma forma transparente na visão do Processador Central. A arquitetura do Controlador de Rede aqui sugerida é ilustrada na Figura 14.

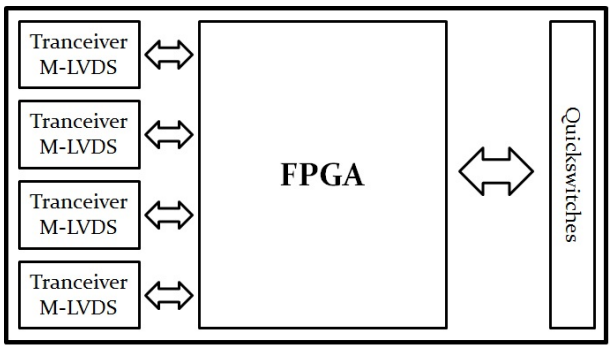


Figura 14: Arquitetura do Controlador de Rede

2.2.5 Controle Embarcado

O Controle Embarcado deve ser um sistema simples porém muito confiável. Basicamente este deve consistir de um processador local, alguns botões e de uma interface visual, como um *display* LCD . O Controle Embarcado deve se comunicar com o Processador Principal através de uma interface serial RS-232 e para tal pode ser utilizado um conversor de nível LVTTL para RS-232. Quando o Processador Central é ligado, um programa deve automaticamente ser executado após a inicialização do sistema operacional. Este deve ciclicamente requisitar dados do controle checando assim as configurações atuais do usuário, as quais devem ficar retidas na memória do processador local. Este, por sua vez, é aconselhável que seja um microcontrolador. Uma visão de uma arquitetura sugerida do Controle Embarcado é ilustrada na Figura 15.

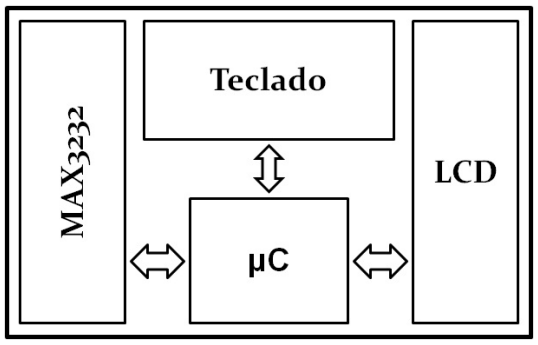


Figura 15: Arquitetura do Controle Embarcado

3 IMPLEMENTAÇÃO

Neste capítulo é descrita a implementação do sistema projetado, apresentando aspectos relevantes das camadas de hardware e de software desenvolvidas. Para finalizar é apresentada uma breve discussão sobre o resultado obtido.

3.1 Implementação do Hardware

Nesta seção será detalhada a implementação hardware do sistema projetado. É importante ressaltar que o controle embarcado não foi incluído pois as suas características físicas (formato, dimensões e tamanho da tela) serão definidas em um ponto mais avançado do projeto.

3.1.1 Processador Central

Como já comentado na Capítulo 2, foi definido o formato PC/104+ para o processador central. Existem várias opções comerciais disponíveis neste formato de computador baseados em diferentes processadores. De forma a simplificar a compatibilidade com softwares disponíveis para computadores convencionais ficou definido usar um processador Intel® Atom™. Além da compatibilidade, o processador em questão foi considerado o dispositivo disponível no mercado com maior capacidade de processamento sem a necessidade

do uso de uma ventoinha para refrigeração. Isto é de extrema importância pois reduz a quantidade de partes móveis no sistema, aumentado assim a confiabilidade do mesmo. Desta forma, a placa PCM-3362N da empresa Advantech foi escolhida para ser o Processador Central do sistema desenvolvido. Ela tem embarcado um processador Intel Atom® N450, com uma frequência máxima de processamento de 1,67 Ghz, em conjunto com 2 GB de memória flash e uma memória RAM DDR2 de 1GB. Para finalizar, ele tem disponível uma interface Gigabit Ethernet, duas portas RS-232 e quatro portas USB2.0, assim atendendo a todos os requisitos de projeto. Como será descrito mais adiante, a PCM-3362N ainda não foi implementada como Processador Central, porém já se encontra em posse do grupo. Uma foto da placa adquirida pode ser vista na Figura 16.

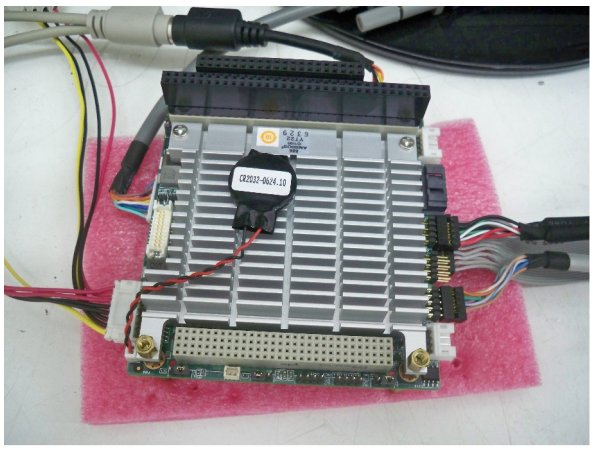


Figura 16: Foto da placa PCM-3362N

3.1.2 Controlador de Rede

3.1.2.1 Placa

Apesar do projeto da placa do controlador de rede ser relativamente simples, a sua implementação requer uma série de cuidados devido aos requisitos restritos do barramento PCI. Este projeto requer a utilização de placas de cir-

cuito impresso com pelo menos 4 camadas, o que é relativamente caro de ser fabricado. Deste modo, o autor realizou todo o desenvolvimento deste trabalho utilizando uma placa PCI com um formato convencional, o que restringiu a implementação do sistema para testes a um computador *desktop*. A placa utilizada, chamada Raggedstone1, foi comprada da empresa Enterpoint. Ela consiste de um FPGA Spartan® - 3 da Xilinx®, um circuito de conversão de nível do barramento PCI(5V) para LVTTL(3.3V) e os pinos disponíveis no FPGA roteados para diversas barras de pinos. O computador utilizado, cujo processador é um Pentium® III, foi reaproveitado de um lote que seria descartado. Uma foto da placa conectada ao computador utilizado pode ser visto na Figura 17.

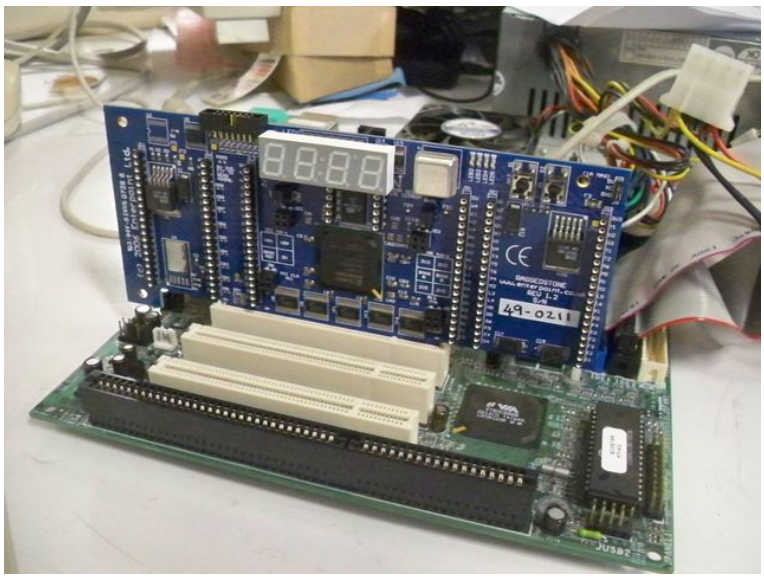


Figura 17: Raggedstone1 conectada ao computador

Uma placa de expansão com tranceivers M-LVDS foi desenvolvida para ser utilizada com a Raggedstone1. Foram utilizados os circuitos integrados SN65MLVD204A (TI, 2003b), os quais são tranceivers do tipo 2, tendo assim um *offset* positivo no limiar de transição entre os níveis lógicos 0 e 1. Esta característica permite que o sistema tenha um nível lógico definido na ausência

de sinal em seus terminais, o que é de extrema importância em comunicações *half-duplex*. Porém, este mecanismo se mostrou insuficiente, sendo falho mediante à presença de pequenas oscilações causadas pela transição abrupta do sinal de um nível negativo para alta impedância (nenhum driver ativo). De forma a solucionar tal problema, foram adicionados resistores de 390 ohms como mostrado na Figura 18, de forma a criar um nível de tensão bem definido no sistema quando nenhum driver se encontra ativo (GINGERICH, 2002). O resistor de 100 ohms utilizado foi escolhido de forma a realizar o casamento de impedância com o cabo utilizado.

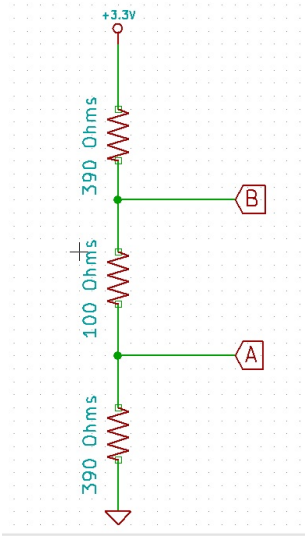


Figura 18: Circuito de *offset*

Foram utilizados conectores RJ45 em conjunto com cabos CAT5e flexíveis convencionais para as conexões da rede. Estes foram escolhidos por atenderem o requisito de taxa de transmissão definida e por terem uma impedância diferencial bem controlada. Além disso, os 4 pares de fios disponíveis em um mesmo cabo possibilitam tanto a transmissão dos sinais de comunicação, como a alimentação dos circuitos digitais, reduzindo a quantidade de cabos cruzando o sistema.

3.1.2.2 Lógica do FPGA

Como sugerido no Capítulo 2, a implementação da lógica interna do FPGA foi feita de uma forma modular possibilitando uma fácil portabilidade da lógica do Controlador de Rede desenvolvido. Aqui serão apresentados detalhes sobre a implementação de cada um dos seguintes três módulos: interface PCI, interface Wishbone, Controlador de Rede. Uma ilustração da organização da lógica interna do FPGA pode ser vista na Figura 19.

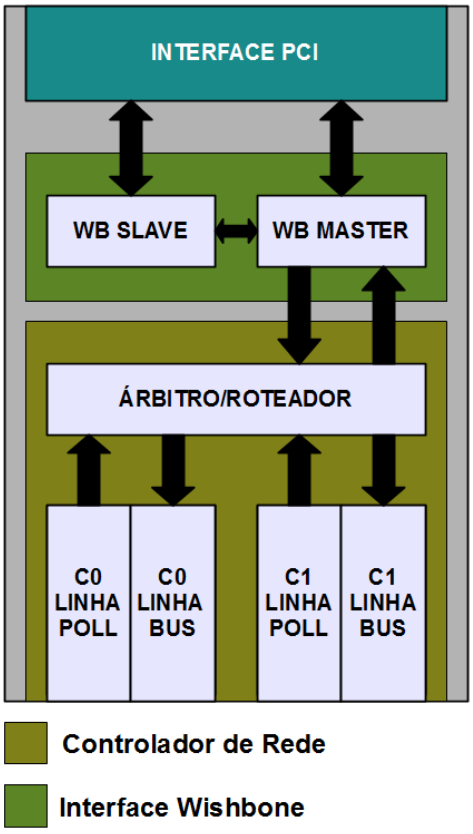


Figura 19: Lógica do FPGA do Controlador de Rede

I - PCI

Foi utilizado um controlador PCI baseado no PCI Bridge IP Core, desenvolvido por Miha Dolenc e Tadej Markovic (DOLENC; MARKOVIC, 2002). Este

foi escolhido por possibilitar o uso do controlador como mestre do barramento PCI. Por motivos técnicos aqui não relevantes, diversas arquiteturas de computador atuais não têm um controlador de *Direct Memory Access* (DMA) disponível para o barramento PCI, desta forma não permitindo que o mesmo realize transmissões em modo *burst*. Este fato reduz drasticamente a banda de transmissão do barramento. Porém, esta modalidade de transmissão é possível caso seja inicializada pela placa conectada ao barramento PCI utilizada e para tal a mesma tem que se tornar mestre do barramento.

O modo que o módulo disponibiliza para realizar a conexão da lógica interna do FPGA com o barramento PCI é através de duas interfaces Wishbone independentes: uma mestre e outra escrava. A primeira é utilizada para passar informações recebidas pelo módulo quando o mesmo está operando no barramento PCI no modo escravo. De uma maneira complementar, a segunda é utilizada de forma a controlar o módulo quando se deseja que o mesmo atue como um mestre do barramento PCI. Para finalizar, o IP Core em questão implementa duas FIFOs de forma a sincronizar os domínios de clock do barramento PCI com o da lógica interna do FPGA. Tal mecanismo possibilita a utilização de frequências de clock arbitrárias no sistema.

II - Controlador de Rede

O Controlador de Rede desenvolvido é constituído de três sub-módulos independentes: Canal de Rede, Árbitro e Roteador.

Um módulo do Controlador de Rede pode conter até oito Canais de Rede. Estes são sub-módulos idênticos responsáveis por gerenciar, de forma independente, os diferentes canais existentes no sistema. Os canais, por sua vez, são compostos por um controlador da Linha Poll e outro controlador da Linha

Bus. Cada controlador tem uma FIFO própria de forma a acomodar as diferenças de taxa de transmissão entre a lógica interna do FPGA e das linhas de comunicação externas. Ambos os controladores implementam a checagem de CRC das comunicações, em conjunto com o controle de fluxo e arbitragem de cada linha. Além disso eles possuem um temporizador de forma a controlar o *timeout* das mensagens. Para finalizar, o controlador da Linha Poll modifica o valor do campo de endereço de todas as mensagens válidas recebidas, escrevendo no mesmo o valor do endereço do Nó que a enviou. Este procedimento é realizado de forma a trocar informações irrelevantes para o Processador Central por dados que possam ter utilidade para a sua operação.

Para realizar a recuperação dos dados recebidos pela rede, foi utilizado o método de *oversampling* com clocks defasados como descrito em (SAWYER, 2009). Este método utiliza dois sinais de clock defasados em 90 graus, fazendo uso tanto as bordas de subida quando as de descida para amostrar os dados recebidos. Desta forma, se consegue uma taxa de amostragem quatro vezes maior que o próprio clock utilizado. A Figura 20 (extraída de (SAWYER, 2009)) ilustra tal procedimento.

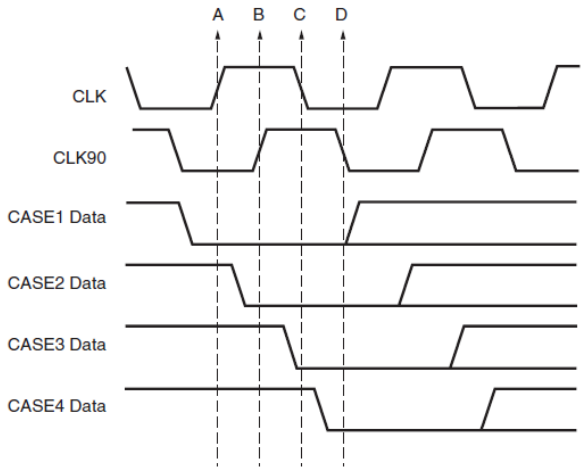


Figura 20: *Oversampling* com dois cloks defasados

O Árbitro é responsável por gerenciar o envio das mensagens recebidas

pela rede para o Processador Central. O sistema de arbitragem empregado é baseado no princípio de que as mensagens provindas de canais de um número menor têm preferência. Apesar de não ser um mecanismo de arbitragem justo, atende às necessidades do sistema já que uma lógica mais complexa não traria nenhum benefício adicional. De modo a adicionar informações relativas à mensagem para o Processador Central, o Árbitro troca os três bits mais significativos do campo de endereço de destino da mensagem pelo número do canal de origem da mesma.

Para finalizar, o Roteador é responsável por distribuir as mensagens recebidas pelo computador entre os canais de acordo com o endereço das mensagens. Como já explicado anteriormente, os três bits mais significativos no byte relativo ao endereço da mensagem representam o canal em que o nó endereçado se encontra. Já que está informação não é relevante após o roteamento, os três bits são preenchidos com o valor lógico zero.

III - Interface Wishbone

A interface Wishbone foi desenvolvida de forma a criar uma lógica para gerenciamento da rede e controle de fluxo de dados. Ela é formada por dois sub-módulos: Wishbone Slave e Wishbone Master. O primeiro é utilizado como uma interface do Controlador de Rede para o Processador Central e fica conectado à interface Wishbone mestre do módulo do controlador PCI. Ele implementa um série de registradores de 32 bits, os quais são utilizados tanto para configurar e controlar a rede, quanto para disponibilizar uma série de informações para o Processador Central. Um mapa completo desses registradores assim como uma breve descrição dos mesmos pode ser visto no Apêndice A. Os registradores mais importantes para o funcionamento do sistema de comunicação desenvolvido são: READ_COMMAND, INTERRUPT_STATUS,

TX_BASE_ADDRESS, RX_BASE_ADDRESS e RING_BUFFER_STATUS.

O Wishbone Master, por sua vez, serve para controlar o módulo PCI quando este está atuando como mestre do barramento. As suas operações de leitura e escrita na interface Wishbone escravo do controlador PCI refletem as mesmas operações no barramento PCI. No Processador Central devem ser alocadas duas regiões da memória para realizar a troca de dados com a rede. A primeira delas deve conter dois buffers de envio de dados posicionados de uma forma sequencial. Estes são chamados de TxBuffers. O Processador Central deve então escrever no registrador TX_BASE_ADDRESS o endereço do começo desta região da memória.

De modo a enviar mensagens para rede, o Processador Central deve proceder da seguinte maneira. Primeiro ele deve escrever todas as mensagens a serem enviadas alinhadas em um dos TxBuffers. Depois ele deve escrever nos 29 bits menos significativos do registrador READ_COMMAND a quantidade de DWORDs que as mensagens escritas ocupam na memória e nos dois bits remanescentes em qual buffer que se encontram as mensagens. O sub-módulo Wishbone Slave então irá passar tais informações para o Wishbone Master e este, por sua vez, irá realizar a leitura de todas as mensagens na memória atuando como mestre do barramento PCI, para então repassá-las para o Roteador. Quando a leitura estiver terminada, uma interrupção é gerada.

A segunda região de memória alocada deve conter 10 buffers de recepção de dados alinhados, os quais são chamados de RxBuffers. O Processador Central deve escrever o endereço do início de tal região no registrador RX_BASE_ADDRESS. Quando existem dados disponíveis nas FIFOs dos canais do Controlador de Rede, o sub-módulo Wishbone Master escreve todas as mensagens armazenadas de uma forma sequencial em um dos 10 RxBuf-

fers que esteja disponível, inicializando esta escrita na segunda DWORD do referido buffer. A disponibilidade ou não destes buffers é descrita pelo registrador RING_BUFFER_STATUS. Quando todos as FIFOs se encontram vazias, ele escreve a quantidade de DWORDs enviadas nos primeiros 32 bits do Rx-Buffer utilizado. Feito isso ele gera uma interrupção e marca o buffer como inativo no registrador RING_BUFFER_STATUS.

Todas as interrupções geradas são descritas pelo registrador INTERRUPT_STATUS. Qualquer leitura ao referido registrador faz com que o mesmo tenha todos os seus bits colocados em nível lógico 0. De modo a liberar os buffers inativos, o Processador Central deve realizar uma escrita no registrador RING_BUFFER_STATUS como descrito no Apêndice A.

3.1.3 Nós

3.1.3.1 Placa

Para simplificar a implementação deste sistema em diferentes aplicações, a arquitetura dos Nós foi dividida em duas placas diferentes. A primeira, chamada de placa-mãe, contém todo o hardware relativo à interface de rede, em conjunto com o processador local. A segunda, chamada placa-filha, contém todo o hardware específico para realizar a interface com sensores e atuadores. Este formato cria um grau extra de flexibilidade pois permite que uma mesma placa-mãe seja utilizada em diversas aplicações diferentes, cada uma com uma placa-filha dedicada.

Para implementar à interface de rede do Nó foi utilizado um FPGA Spartan®-3AN da Xilinx® com 1584 *Logic Elements* (LE) e um encapsulamento do tipo *Low-profile Quad Flat Pakcage* (LQFP) de 144 pinos (XILINX, 2011). Este foi escolhido por ser o menor (menos recursos lógicos) FPGA

encontrado com um *Phase Lock Loop* (PLL) disponível. Este, por sua vez, é necessário para gerar as frequências internas utilizadas na lógica de controle e transmissão de dados. Além disso, o mesmo contém uma memória flash para configuração do FPGA dentro do mesmo encapsulamento, simplificando o layout da placa. Em uma próxima versão provavelmente será utilizado um XC3S50A (Spartan®-3A) o qual não é munido de uma flash interna, porém é disponibilizado em um encapsulamento de 100 pinos, o que ocupa uma área de placa menor.

O microcontrolador escolhido para fazer o papel do processador local foi o PK40N512VLQ100 (FREESCALE, 2011) da Freescale. Ele é baseado no processador ARM Cortex-M4, o qual possui diversas instruções dedicadas a processamento digital de sinais e processa informações a uma frequência de até 100 Mhz. Este microcontrolador foi escolhido devido à sua grande variedade de periféricos disponíveis, incluindo 24 canais de conversores Analógico-Digital (ADC), dois conversores Digital-Analógico (DAC), quatro canais de *Pulse Width Modulation* (PWM) e duas interfaces para encoder de quadratura. Este conjunto de periféricos pode ser encontrado em outros microcontroladores, porém os grandes diferenciais aqui considerados foram o fato de que a resolução máxima dos canais ADC é de 16 bits, em conjunto com a possibilidade de dois deles poderem ser configurados como conversores diferenciais.

Para finalizar, foram utilizados os tranceivers SN65MLVD204A para implementar a interface em nível físico com as linhas M-LVDS. As trilhas responsáveis por transmitir os sinais diferenciais na placa foram cuidadosamente desenhadas de forma a terem uma impedância diferencial casada com o cabo CAT5e. Uma foto da placa-mãe desenvolvida pode ser visto na Figura 21

Nenhuma placa-filha para uma aplicação real foi desenvolvida até então,

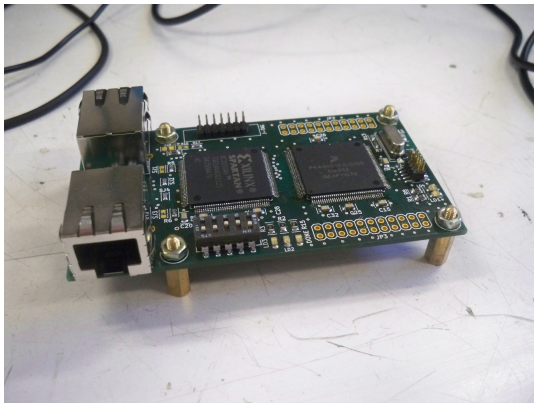


Figura 21: Placa-mãe do Nó desenvolvida

porém um exemplo de aplicação da arquitetura implementada é ilustrado pela Figura 22. Não obstante uma placa simplória foi utilizada na caracterização da rede, a mesma não possui muita utilidade prática. Esta, por sua vez, é descrita no capítulo 4 mais detalhadamente.

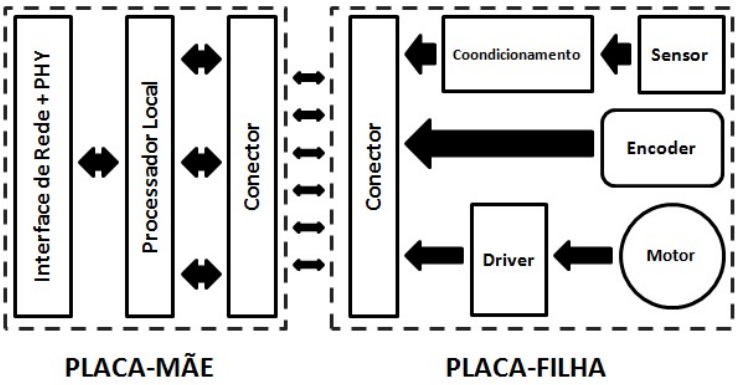


Figura 22: Arquitetura do Nó implementado

3.1.3.2 Lógica do FPGA

De uma forma similar ao Controlador de Rede, a lógica interna do FPGA da interface de rede dos Nós foi desenvolvida de uma maneira modular. A parte do controle lógico da rede é composta por 3 sub-módulos: Controlador da Linha Poll, Controlador da Linha Bus e o Gerente de Mensagens. Os dois primeiros implementam a lógica necessária para realizar a interface com a

rede, incluindo a checagem de CRC e o gerenciamento de mensagens duplicadas. Além disso, o Controlador da Linha Bus remove o byte referente ao endereço da mensagem antes de repassá-la ao processador local, de forma a diminuir a quantidade de dados a serem lidos. De uma forma similar, o submódulo controlador da Linha Poll adiciona o endereço do Controlador de Rede em toda mensagem enviada para rede. O método utilizado pela interface de rede dos Nós para realizar a recuperação de dados é o mesmo empregado no Controlador de Rede.

Seguindo a mesma lógica utilizada do Controlador de Rede, tanto o Controlador da Linha Bus, como o da Linha Poll possuem uma FIFO entre eles e a interface com o processador local, de forma a acomodar as diferenças de taxa de transmissão. Porém, no caso da Linha Poll, tal solução é insuficiente visto que existe a possibilidade de a interface de rede ter uma taxa de transmissão inferior à da rede de comunicação, sendo assim necessário um Gerenciador de Mensagens. Este, por sua vez, controla o número de mensagens completas que se encontram armazenadas na FIFO, repassando tal informação para o Controlador da Linha Poll. Este mecanismo não tem um impacto relevante na Linha Bus pelo fato de que, após serem verificadas como válidas, as mensagens são transferidas para a FIFO a uma taxa de transmissão de 400 Mbps, cujo valor foi considerado razoável para ser posto como um limite a ser utilizado.

Para finalizar, a interface de comunicação com o processador local é totalmente dependente do dispositivo utilizado. Nesta implementação foi utilizada uma interface *Serial Peripheral Interface* (SPI) por ser disponível na grande maioria dos microcontroladores. Além disso é relativamente simples de ser utilizada e normalmente é uma das interfaces seriais com maior taxa de transmissão disponível. A Figura 23 ilustra a lógica interna do FPGA implementada.

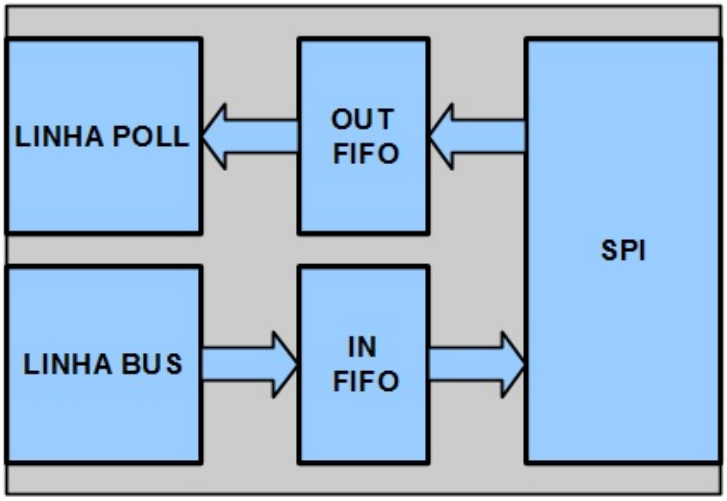


Figura 23: Lógica do FPGA da interface de rede dos Nós

3.1.3.3 Firmware do Processador Local

De forma a abstrair a troca de informações com os Nós foram implementados o que foi chamado de *soft-registers*. Estes, por sua vez, são emulações de registradores criados pelo *firmware* dos processadores locais. Cada Nó tem um mapa de *soft-registers* próprios e esses são divididos em três conjuntos: *read registers*, *write registers* e *config registers*. O primeiro conjunto de registradores carregam informações que podem ser lidas pelo Processador Central, sendo eles majoritariamente dados provindo de sensores. Ja os *write registers* possuem dados cujos valores são escritos pelo Processador Central, porém sempre relevantes às rotinas de controle dos mesmos. O conjunto de *config registers* também carregam informações enviadas pelo Processador Central, porém sempre relacionados a alguma configuração interna do Nó em questão. Este último, apesar de já definido, não foi implementado no Nó desenvolvido. Cada *soft-register* podem ter um tamanho arbitrário de acordo com os dados que eles carregam, porém não podendo exceder 32 bits. Os conjuntos de *soft-registers* pode ter um tamanho total máximo de 255 bytes cada.

Na atual implementação foram criados quatros comandos para realizar a interface com os Nós: GET, SET, EXCHANGE e NODE_CONFIG. O comando GET é utilizado para o Processador Central realizar solicitações de leitura dos dados armazenados nos *read registers* dos Nós. Mensagens que utilizem o comando GET devem ter um campo de dados obrigatoriamente com tamanho de zero bytes. O comando SET, por sua vez, é utilizado tanto para o Processador Central escrever dados para os *write registers* dos Nós, como para a resposta dos Nós relativa a um comando GET ou EXCHANGE recebido, enviando assim as informações armazenadas nos *read registers*. Em ambos os casos, os dados do conjunto de *soft-registers* utilizados devem ser dispostos alinhados no campo de dados da mensagem em questão. Desta forma, um byte dentro deste campo pode conter dados relativos a mais de um *soft-register*. A Figura 24 exemplifica o campo de Dados de um comando SET recebido por um Nó que tenha um conjunto de três *write registers*, cada um com um tamanho de 12 bits.

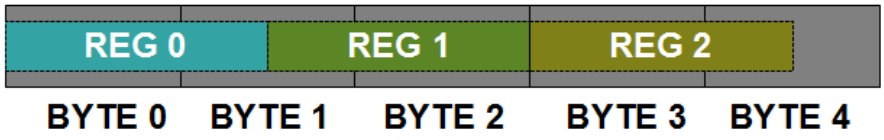


Figura 24: Exemplo de campo de dados gerado por uma mensagem do tipo SET

O comando EXCHANGE é uma junção do comando SET com o comando GET. Ele só pode ser enviado pelo Processador Central e tem um formato de mensagem idêntico a mensagens com comando SET, porém com a única diferença de que ele também solicita a leitura dos *read registers* do Nó endereçado. Para finalizar, o comando NODE_CONFIG, o qual também só pode ser enviado pelo Processador Central, tem uma forma de funcionamento similar ao comando SET, porém com a diferença de que os dados carregados pela mensagem são relativos aos *config registers*.

3.2 Implementação do Software

Todo o trabalho foi implementado utilizado a distribuição Debian do Linux versão 2.6.32-5. Apesar de que durante a fase de projeto ficou definida a utilização do *patch* Xenomai, este ainda não foi implementado. Na arquitetura do Linux, o acesso ao hardware é feito através de kernel drivers. Desta forma um driver dedicado ao Controlador de Rede implementado foi desenvolvido. De forma a facilitar o uso do mesmo, um conjunto de interfaces para o *User Space* foram implementadas.

3.2.1 Kernel Driver

Aqui não serão dadas explicações sobre o funcionamento de um driver para um dispositivo PCI no Linux. Caso o leitor esteja interessado no assunto as seguintes referências foram utilizadas durante o desenvolver deste trabalho: (VENKATESWARAN, 2008; CORBET et al., 2005). As rotinas padrões de inicialização do driver não serão aqui detalhadas e nem a sua estrutura de forma geral. Serão abordados apenas os pontos específicos relativos ao funcionamento do sistema em questão.

Inicialmente, foi utilizada uma adaptação do driver desenvolvido por Paolo Prete (PRETE, s.d.) de forma a realizar a interface com o Controlador de Rede. Porém, o mesmo se mostrou insuficiente para o sistema pois não possuía a lógica necessária para realizar transmissões de dados com a placa PCI atuando como mestre do barramento. Além disso, não havia nenhum código relativo ao gerenciamento de interrupções implementado. Desta forma um novo driver foi escrito cobrindo tais déficits.

Após a rotina de inicialização padrão do driver, a primeira coisa que o mesmo faz é alocar na memória espaços para os TxBuffers e RxBuffers

utilizando a função **pci_alloc_consistent**. Uma vez alocados, o driver escreve os endereços de tais buffers nos registradores TX_BASE_ADDRESS e RX_BASE_ADDRESS respectivamente. Feito isso, o driver fica inativo até que o mesmo seja aberto pela aplicação no *User Space*. Quando isto acontece, o driver requisita da kernel o valor da linha de interrupção alocado para o Controlador de Rede habilitando o seu uso.

Além do intervalo de memória alocado disponível para o Controlador de Rede, cada região de memória contém 4 bytes extras utilizados apenas como uma interface com o *User Space*. Esses bytes estão posicionados no primeiro endereço das regiões de memórias alocadas e são chamados de TX_BUFFER_FLAGS e RX_BUFFER_FLAGS. O primeiro deles é utilizado para a aplicação ter controle de quais TxBuffers estão disponíveis para serem utilizados. Os dois bits menos significativos representam o estado dos dois TxBuffers implementados. Caso eles estejam em nível lógico 1 o buffer em questão está livre. De uma forma similar, o RX_BUFFER_FLAGS é utilizado para sinalizar quando um RxBuffer se encontra com dados para leitura. Da mesma forma que o TX_BUFFER_FLAGS, cada um dos 10 bits menos significativos representa o estado de um dos dez buffers existentes. Um nível lógico 1 representa que o RxBuffer em questão possui dados pendentes para leitura.

O driver oferece alguns serviços através de uma interface do tipo *char driver*, disponibilizando as seguintes *file operations*: *write*, *read*, *open*, *release*, ioctl, llseek e mmap. As duas primeiras são utilizadas para realizar operações de escritas e leitura respectivamente do dispositivo PCI . As operações *open* e *release* são utilizadas respectivamente para abrir e fechar a interface com o *char driver*. A ioctl oferece algumas configurações do driver não relevantes a esta descrição. A operação llseek é utilizada para modificar os endereços de escrita e leitura das operações *write* e *read* respectivamente. Para finali-

zar, a operação mmap é utilizada para mapear os buffers de entrada e saída de dados na memória do *User Space*, dando assim acesso direto a esses endereços para a aplicação.

O driver também implementa uma função para ser chamada mediante o evento de uma interrupção gerada pelo Controlador de Rede. Quando isto acontece, uma leitura ao registrador INTERRUPT_STATUS é feita de forma a determinar os motivos que geraram tal sinal de interrupção. Feito isso, a função atualiza os valores da RX_BUFFER_FLAGS e TX_BUFFER_FLAGS de acordo com as informações recebidas. Na atual implementação, o gerenciamento das interrupções geradas por erros de transmissão na rede não foi incluído.

3.2.2 User Space

A implementação em *User Space* foi dividida em três graus de abstração: *Application Programming Interface* (API), *Manager* e Aplicação. Os dois primeiros fazem parte do projeto do sistema desenvolvido neste trabalho e o terceiro é relacionado à aplicação em si. Aqui será dado um breve resumo do *Manager* e da API, porém o conjunto completo de funções disponíveis assim como uma breve descrição das mesmas pode ser visto no Apêndice B. É importante ressaltar que as bibliotecas foram criadas utilizando a linguagem de programação C.

A API serve como uma interface de baixo nível para o driver, lidando com as funções de comunicação com o *char driver* disponíveis. De modo geral, é utilizada uma estrutura de dados denominada **rb_desc_t** para descrever a conexão com o Controlador de Rede. Esta estrutura carrega o endereço para os buffers de entrada e saída de dados em conjunto com a variável que representa a conexão com o driver. Só deve haver uma instância desta estrutura na

memória, já que a mesma é uma abstração do Controlador de Rede na visão do *User Space*. Dentro da API essa instância é denominada **r_bone** e assim será chamada neste trabalho. Para inicializar a interface, a função **rb_init** deve ser chamada. Esta, por sua vez, é responsável por alocar na memória o espaço da estrutura **r_bone**, abrir a conexão com o driver e mapear os buffers alocados pelo driver para o *User Space* utilizando **mmap**. O retorno da função **rb_init** é o endereço de **r_bone**.

Com a interface inicializada, a API fornece duas funções principais para realizar leitura e escritas nos registradores do Controlador de Rede: **rb_read** e **rb_write**. Essas duas rotinas abstraem as funções **read** e **write** respectivamente. Baseadas nelas, outras funções foram criadas de forma a deixar a interface com a rede mais intuitiva. Um exemplo destas rotinas é a **rb_command_read**, cuja funcionalidade é escrever no registrador COMMAND_READ com o formato correto. Para finalizar, a API fornece uma função chamada **rb_close** para fechar todas as conexões e liberar os espaços de memória alocados.

Criando mais um nível de abstração em cima da API, foi desenvolvida a biblioteca *Manager*. Esta, por sua vez, foi criada para facilitar o gerenciamento de troca de mensagens com os Nós. Ela é baseada em uma série de estruturas de dados, sendo **node_t**, **node_list_t**, **request_desc_t** e **rb_mngr_t** as quatro mais importantes. A primeira é utilizada como uma imagem dos Nós dentro da aplicação, a qual contém um descritivo dos *soft-registers* em conjunto com ponteiros para as variáveis representativas dos mesmos. De forma a simplificar a utilização desta estrutura, foi criada uma rotina chamada **rb_mk_node**, a qual aloca dinamicamente na memória o espaço para o Nó em questão baseado em um *template*. Desta forma, Nós desenvolvidos por outros grupos e empresas podem ser utilizados sem nenhum conhecimento

profundo sobre o mesmo, desde que seja fornecido o tal *template* em conjunto com o descritivo da funcionalidade de cada *soft-register.*

A estrutura **node_list_t** é utilizada para criar uma descrição das trocas de informações com os Nós. Deve ser criado um vetor baseado nesta estrutura, sendo cada índice do mesmo a descrição de uma transação a ser realizada com um Nó especifico. Na estrutura devem ser armazenados o ponteiro para a estrutura do tipo **node_t** representativa do Nó com que será realizada a troca de dados, em conjunto com as *flags* que representam o tipo de transação a ser realizada. Na implementação atual foram definidas apenas a READ_FLAG e a WRITE_FLAG. Quando apenas a primeira é utilizada, é gerada uma transação do tipo GET. De uma forma similar, quando apenas a segunda é utilizada é gerada uma transação do tipo SET. Para finalizar, quando ambas são utilizadas é gerada uma transação do tipo EXCHANGE.

Como já definido anteriormente no capítulo 2, o campo de ID tem como função principal servir como um identificador para o Processador Central controlar as suas solicitações enviadas para os Nós. De forma a auxiliar tal processo, foi criada a estrutura de dados **request_desc_t**, a qual armazena um descritivo das solicitações enviadas. Ela é composta por uma *flag* que representa se o ID em questão já está sendo utilizado, em conjunto com um ponteiro para a estrutura **node_t** relacionada ao Nó com que a transação está sendo feita. Para a sua utilização deve ser criado um vetor contendo 256 estruturas do tipo **request_desc_t**, sendo que o índice do vetor representa o número de ID utilizado para as transações. Nesta tabela devem ser armazenadas apenas solicitações que tenham uma resposta dos Nós, ou seja, os comandos GET e EXCHANGE na atual implementação.

Para finalizar, a estrutura do tipo **rb_mngr_t** foi criada para simplificar o processo de gerenciamento da rede como um todo. Esta é composta pela ta-

bela formada pelo vetor dos 256 **request_desc_t**, pelo ponteiro para o **r_bone** e pelo ponteiro para o vetor formado pelas estruturas do tipo **node_list_t**. De forma a simplificar o seu uso, foram criadas quatro funções de gerenciamento que recebem como argumento apenas o ponteiro para a estrutura em questão. Estas são: **mngr_init**, **mngr_chk_req**, **mngr_wt_buff** e **mngr_buff_parse**.

A função **mngr_init** foi criada para simplificar a inicialização de todo o sistema. Para tal ela chama a função **rb_init**, inicializa a tabela de IDs e coloca os TX_BUFFER_FLAGS e RX_BUFFER_FLAGS em um estado conhecido. A função **mngr_chk_req**, por sua vez, é uma rotina que checa se existe alguma solicitação pendente na rede. A função **mngr_wt_buff** é utilizada para montar o *stream* de mensagens, representado pelo vetor de estruturas do tipo **node_list_t**, em um TxBuffer livre. Feito isto, a mesma função então "manda" o Controlador de Rede ler o referido buffer utilizando a função **rb_command_read**.

Para finalizar, a função **mngr_buff_parse** inicialmente verifica se existem pendências nos RxBuffers. Caso isso seja verdade, ela automaticamente interpreta os dados recebidos distribuindo as informações para as variáveis correspondentes, utilizando as estruturas de dados descritas acima como referência. Ela finaliza o processo escrevendo no registrador RING_BUFFER_STATUS do Controlador de Rede, liberando os buffers lidos.

3.3 Discussão

Como resultado do sistema implementado é possível avaliar a abstração criada, tanto na visão de um desenvolvedor de Nós, quanto para um usuário trabalhando com uma aplicação no Processador Central. Para este último, a interface com a rede pode ser resumida como uma série de Nós virtuais, cujos

soft-registers são mapeados diretamente para variáveis utilizadas na rotina de controle. De forma a atualizar tais Nós virtuais com os dados armazenados nos Nós reais, é utilizada uma estrutura chamada **rb_mngr_t**, a qual oferece quatro serviços de simples uso como interface.

Pode-se ver então, que toda a complicação relacionada à temporização, checagem de dados, endereçamento e controle de fluxo de dados foi abstraída, criando assim uma interface cuja implementação requer pouco esforço e conhecimento técnico, sem abdicar de uma resposta determinística e confiável. Tal abstração é ilustrada pela Figura 25.

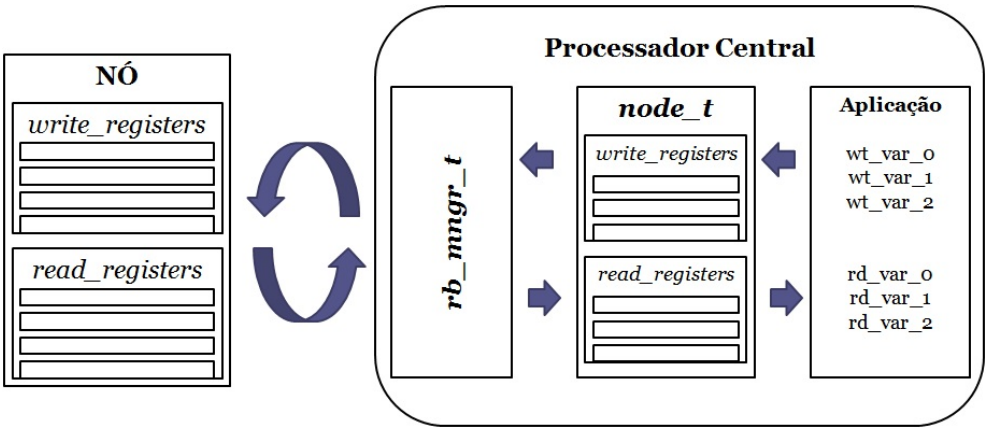


Figura 25: Abstração da interface no Processador Central

Complementarmente, na visão de um desenvolvedor de Nós, a interface com o Processador Central é resumida à duas FIFOs de entrada e saída de dados, cuja interface é SPI. A interpretação das mensagens pode ser realizada com uma simples máquina de estados (FSM) em conjunto com uma operação *switch/case* para lidar com os possíveis comandos recebidos. O mapeamento dos *soft-registers* pode ser implementado apenas com a utilização de operações de *bitshift* e *bitwise*, já que a estrutura dos registradores é fixa independentemente da aplicação.

A montagem das mensagens de resposta é igualmente simples de ser im-

plementada, já que o cabeçalho terá apenas como dado variável o número identificador, cujo valor é fornecido pela solicitação recebida. Adicionalmente à abstração criada para lidar com toda complicação de se conseguir uma comunicação determinística e confiável, há um desacoplamento do desempenho do Nó desenvolvido com o comportamento geral da rede. Desta forma, não há um compromisso de o desenvolvedor em questão ter uma vasta experiência em programação de forma a conseguir uma resposta otimizada. A abstração descrita é ilustrada pela Figura 26.

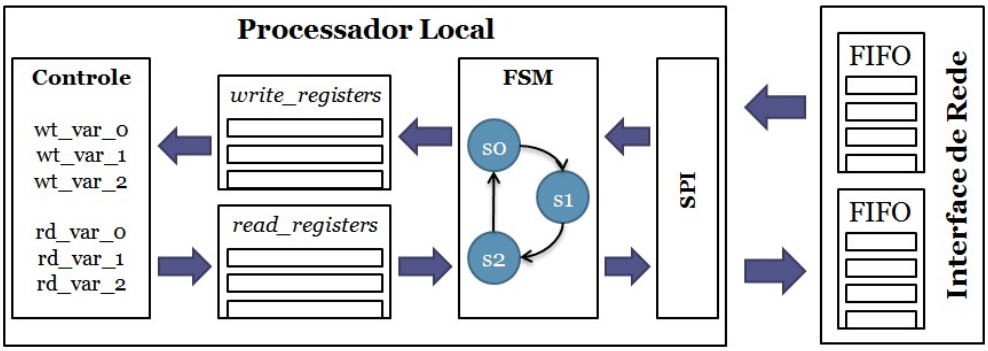


Figura 26: Abstração da interface nos Nós

4 TESTES

Foram realizados dois testes de forma a caracterizar o sistema de controle desenvolvido nos quesitos latência e *skew*. A seguir é dada uma breve descrição do circuito utilizado nos testes, para então detalhar os resultados obtidos.

4.1 Hardware de Teste

Uma placa simplória foi desenvolvida de forma incluir o hardware necessário para criar uma conexão do processador local dos Nós com um gerador de funções. Esta, por sua vez, foi baseada no Circuito Integrado LM358 o qual possui dois amplificadores operacionais em um mesmo encapsulamento, os quais foram colocados no modo seguidor de tensão. A saída de cada um dos amplificadores operacionais foi ligada a um dos canais de um dos ADCs de dois PK40N512VLQ100 e suas entradas não-inversoras foram conectadas ao gerador de funções. Foram utilizados um total de seis Nós para a realização dos testes. Em todos os casos, cada Nó possuía um total de doze *read registers* de 16 bits cada e três *write registers* de 16 bits cada.

4.2 Testes Realizados

4.2.1 Latência

Para realizar o teste de latência do sistema foi preparado um programa de forma a medir o tempo de escrita e leitura de todos os *write* e *read registers* respectivamente dos Nós presentes na rede. Para tal, um comando EXCHANGE foi ciclicamente enviado para cada Nó e então medido o tempo que o sistema levou para ter todas as solicitações recebidas em cada ciclo. Dentro desta medida de tempo estava incluído tanto o tempo para montar os buffers de envio, quanto o tempo de interpretação dos buffers de recepção de dados. O teste foi realizado para um canal operante com um, dois e três Nós presentes. Para finalizar foi realizado um teste para dois canais operantes, cada um com três Nós presentes. Em todas as situações foram realizados 900 ciclos completos. Os resultados obtidos podem ser vistos no gráfico da Figura 27. No gráfico em questão os conjuntos de dados estão nomeados como **xC - yN**, onde x representa a quantidade de canais operantes e y o número total de Nós presentes em todos os canais.

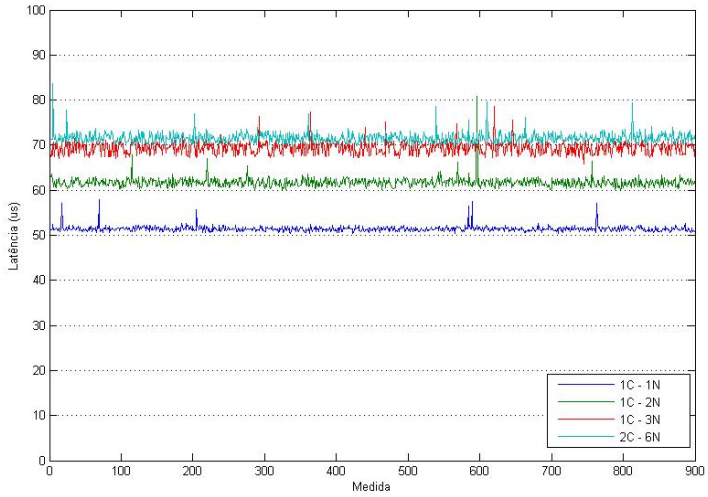


Figura 27: Dados coletado para o teste de latência

O cálculo do valor da latência de um comando EXCHANGE para um dado cenário não é simples, por existirem variáveis incertas no sistema. Porém, para uma estimativa conservadora a seguinte equação pode ser usada:

$$L_t = L_{pc} + T_{cr} + T_t + L_n \tag{4.1}$$

Sendo:

L_{pc}: Latência incluída pelo Processador Central.

T_{cr}: Tempo requerido pelo Controlador de Rede.

T_t: Tempo total de transmissão de todos os dados.

L_n: Latência total relativa a um Nó.

O valor de L_{pc} não pode ser calculado e é dependente de fatores como a frequência de processamento, o sistema operacional utilizado e arquitetura do Processador Central. Este valor deve ser estimado de acordo com o sistema implementado. O T_{cr} no presente caso é predominantemente composto pelo tempo de espera para se tornar mestre do barramento PCI. Para questões de simplicidade foi considerado como o valor fixo de 3 μs. T_t pode ser calculado pela seguinte equação:

$$T_t = n \times ((Q_d + 12) \times 11 + 106) \times 20ns \tag{4.2}$$

Sendo:

Q_d: Quantidade total de dados em bytes trocados com um Nó.

n: Número de Nós presentes em cada canal.

No presente caso Q_d é igual a 30, composto por 24 bytes dos *read registers* e 6 bytes dos *write registers*. À este valor é somado o número 12 relativo aos quatro bytes do cabeçalho e dois bytes do CRC das mensagens enviadas na Linha Poll e Linha Bus. O valor total é multiplicado por 11 devido à estrutura de envio dos bytes descrita no Capítulo 2. O valor 106 é uma composição de diversos fatores. Primeiramente 22 bits relativos ao ACKs enviado como resposta na Linha Bus. Adicionalmente foi somado o valor de 44 bits relativos aos comandos POLL e ACK utilizados na leitura da mensagem na Linha Poll. Para finalizar, foram adicionados 8 bits relativos ao tempo em que as linhas ficam inoperantes devido ao processamento realizado pelas interfaces de rede, totalizando 40 bits. O valor obtido por Nó em cada canal para o presente cenário é de 11,36 μs.

Na equação 4.2 o valor L_n representa a latência incluída por apenas um Nó pois todos processam os pedidos paralelamente, ou seja, apenas o último a receber o comando EXCHANGE que efetivamente terá impacto na latência total. O valor de L_n pode ser calculado por:

$$L_n = T_{pl} + (Q_d + 6) \times 8 \times 40ns + T_{ir} \tag{4.3}$$

Sendo:

$\boldsymbol{T_{ir}}$**:** Tempo requerido pela interface de rede de um Nó.

$\boldsymbol{T_{pl}}$**:** Tempo de processamento do processador local.

T_{ir} é um tempo muito baixo e aqui será considerado como desprezível. No presente caso, o valor de T_{pl} foi medido como aproximadamente 2 μs. O restante do valor de L_n é relativo ao tempo de transmissão entre a interface de rede e o processador local utilizando uma interface SPI a uma taxa de

transmissão de 25 Mbps. Como neste caso não estão presentes o CRC e o endereço da mensagem foi adicionado apenas o número 6 ao valor total de dados transmitidos, referentes ao cabeçalho remanescente das mensagens. Para o presente cenário, L_n foi calculado como 13,520 μs.

Paro o caso de um canal com um Nó presente podemos calcular o valor de L_{pc} como aproximadamente 23 μs. Extrapolando para dois e três Nós teremos o valor total da latência respectivamente de 62 μs e 73 μs. Como pode ser visto na Figura 27, os valores calculados são coerentes com os medidos. Vemos também que para o caso de três Nós, obtivemos um valor inferior com o teste realizado. Isto é decorrente do fato de que os cálculos realizados são conservadores e não levam em consideração a independência de operação das Linhas Poll e Bus.

Observando os gráficos pode-se notar uma boa condição de estabilidade na latência resultante de todos os casos, com exceção de alguns picos espúrios causados pela resposta não determinística do sistema operacional utilizado. Espera-se que estes desapareçam quando utilizado o Xenomai.

Para finalizar, pode-se notar o efeito da utilização de um sistema multicanal, onde ouve um acréscimo de apenas 2 microssegundos no valor médio de latência obtido adicionando-se um canal extra com o mesmo número de Nós. O resultado está de acordo com o esperado e este acréscimo é relativo a diversos pequenos fatores como os tempos de montagem e interpretação dos buffers pelo Processador Central em conjunto com um maior de tempo na transmissão do barramento PCI. Esta característica demonstra o potencial de escalabilidade do sistema onde diferentes partes de um mesmo robô podem ser desenvolvidas separadamente sem se preocupar com a interferência que a mesma terá no sistema como um todo.

Apesar de que os valores obtidos são satisfatórios para se utilizar o sistema desenvolvido com uma malha de controle operando a 4 Khz, acredita-se que o valor de L_{pc} será reduzido com a utilização da PCM-3362N em conjunto com o Xenomai. Estima-se um tempo de aproximadamente 8 μs para L_{pc} possibilitando assim uma latência total de 55 μs para o cenário testado com dois canais. Em uma possível otimização do sistema pode-se reduzir o tempo de L_n trocando a interface de comunicação utilizada com o microcontrolador em troca da perda de flexibilidade. No caso do PK40N512VLQ100 pode-se usar o barramento FlexMemory, com o qual seria possível reduzir o tempo total em aproximadamente 10 μs. Esta opção não foi considerada de princípio pois limitaria seriamente a portabilidade do sistema.

4.2.2 Skew

Idealmente, em um sistema de controle, os dados utilizados em uma rotina devem ser amostrados simultaneamente. No presente trabalho o *skew* é considerado como a defasagem causada nos sinais coletados devido a uma diferença no tempo de amostragem. No presente teste foi gerado um sinal senoidal com frequência de 200Hz com o gerador de funções. Os Nós, cujos ADC estavam conectados ao gerador de funções foram dispostos em diferentes canais da rede. Estes, por sua vez, foram programados de modo a amostrar o sinal gerado a uma taxa de 50Ksps e armazenar os dados em um de seus *read registers*. Foi então criada uma rotina para fazer a leitura de tais dados utilizando o comando EXCHANGE e estes foram salvos em um arquivo, agrupados de acordo com o ciclo de amostragem. O teste foi realizado com um total de três Nós em cada canal. Finalizado o teste, os dados adquiridos foram plotados em um gráfico para analisar a defasagem do sinal amostrado. Adicionalmente foi feita a subtração de ambos os sinais em todos os ciclos. O

resultado pode ser visto na Figura 28.

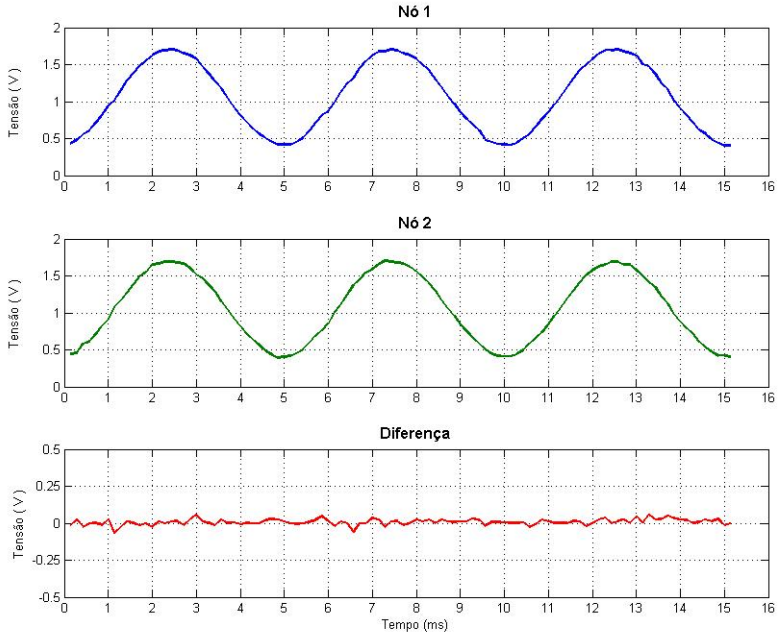


Figura 28: Dados coletado para o teste de *skew*

Com os dados coletados não pode ser notada nenhuma defasagem visível entre os sinais. Caso houvesse uma defasagem constante entre eles o resultado relativo à subtração de ambos deveria resultar em um sinal cossenoidal de mesma frequência. Os valores de diferença obtidos são resultantes de diferença na calibração ADCs utilizados em conjunto com a diferença do ruído presente nos circuitos dos Nós.

5 CONCLUSÕES E TRABALHOS FUTUROS

5.1 Conclusões

Este trabalho apresentou um sistema de sensoriamento e controle hierárquico e distribuído especificamente projetado para ser implementado em sistemas robóticos como exoesqueletos e robôs humanoides. Toda sua arquitetura foi projetada de uma forma modular, composta principalmente por um Processador Central, uma rede de comunicações e diversos Nós distribuídos. Tanto o Processador Central quanto os Nós podem ser trocados no sistema sem nenhuma modificação drástica nas outras partes, desde que sejam seguidas as especificações definidas. Adicionalmente, a arquitetura empregada nos Nós cria um grau extra de flexibilidade no sistema, simplificando o processo de implementação de diferentes sensores e atuadores

Uma rede de comunicação chamada R-Bone foi desenvolvida de forma a atender os requisitos propostos para este trabalho. Esta, por sua vez, opera de uma forma determinista para todas as fontes de erro consideradas. A topologia de barramento, utilizada em conjunto com a implementação de múltiplos canais independentes, possibilita a utilização de apenas um cabo cruzando possíveis membros articulados presentes nos sistemas robóticos. Adicionalmente, tal arquitetura permite uma escalabilidade sem comprometer o desempenho do sistema previamente existente. A utilização de duas linhas de transmissão, com lógica de funcionamento diferente, possibilita um desa-

coplamento da performance dos Nós em relação à resposta do sistema como um todo. Para finalizar, a rede desenvolvida pode ser implementada sem nenhuma configuração adicional durante a inicialização do sistema.

Foi ainda desenvolvido um driver Linux em um conjunto com duas bibliotecas, possibilitando assim uma implementação do sistema em questão sem a necessidade de conhecimentos técnicos avançados sobre o funcionamento do mesmo. O sistema implementado pode ser replicado utilizando componentes comerciais disponíveis em distribuidores de componentes eletrônicos convencionais, simplificando assim o processo de sua utilização em diferentes projetos.

Todas as características supracitadas permitem que a referida rede seja implementada como uma "caixa preta", criando uma camada de abstração para toda a complexidade de se realizar transmissões de uma forma determinista e robusta, sem a necessidade de conhecimentos técnicos avançados para a sua utilização. Isto permite que pesquisadores de diversas áreas possam utilizá-la sem um comprometimento da performance do sistema, além de reduzir o tempo gasto em sua implementação, possibilitando uma maior eficiência dos projetos de pesquisa.

Os testes realizados demonstram que o sistema implementado possibilita a implementação de malhas de controle que tenham como requisito altas frequências de atualização. Foi demonstrado ser possível operar em um sistema com características similares ao apresentado, a uma frequência de 3 Khz com apenas 25% do período de controle utilizado para troca dados. O sistema operou de forma robusta e determinística, com uma latência previsível e controlada.

5.2 Publicações

O presente trabalho teve como resultado a seguinte publicação:

L.F. Rossi, A. Forner-Cordero, and F. Ramirez-Fernandez, "Design of a modular distributed control system for robotic exoskeletons,"in Proceedings of the 3rd Biosignals and Biorobotics Conference, Jan. 2012.

5.3 Trabalhos Futuros

O primeiro ponto a ser tratado futuramente é a fabricação de uma placa com o formato PC/104+ com um FPGA de modo a implementar um Controlador de Rede compatível com a placa PCM-3362N. Além disso, será feito o porte da camada de software desenvolvida para um Linux com um *patch* Xenomai. Quanto à rede de comunicação desenvolvida, será avaliada a inclusão de uma codificação dos bytes transmitidos de forma a deixar o processo de recuperação de dados mais confiável. Uma nova placa-mãe dos Nós será desenvolvida com um formato mais reduzido. Além disso, serão criados pequenos módulos que implementem apenas a interface de rede de forma a flexibilizar a escolha do processador local. Para finalizar, algumas placas-filha serão desenvolvidas para os projetos correntes dos exoesqueletos.

5.4 Comentários Finais

Como já comentado, de forma a simplificar a replicação da rede aqui implementada, todos os códigos fonte foram disponibilizados no servidor SVN com endereço **https://subversion.assembla.com/svn/r-bone/**. Nele se encontram os arquivos de Verilog para a implementação da lógica dos FPGAs do Controlador de Rede e da interface de rede dos Nós. Além disso, também

se encontram disponibilizados os arquivos fonte do driver desenvolvido em conjunto com os arquivos da API e do *Manager*. Todos estão disponíveis sob a licença GNU LGPL (FSF, 2007). Caso o referido servidor não esteja mais disponibilizado, o leitor deve entrar em contato com o Laboratório de Biomecatrônica para obter mais informações.

REFERÊNCIAS

AKACHI, K. et al. Development of humanoid robot hrp-3p. In: *Humanoid Robots, 2005 5th IEEE-RAS International Conference on*. 2005. p. 50 –55.

ALTERA. *Logic Elements and Logic Array Blocks in Cyclone IV Devices*. Nov. 2009.

BARR, M. Programmable logic: What's it to ya? *Embedded Systems Programming*, p. 75–84, Junho 1999.

BETZ, V. Fpga architecture for the challenge. *Internal Report*, s.d.

BOSCH. Can specification. *Version 2.0*, Setembro 1991.

BRUYNINCKX, H. Open robot control software: the orocos project. In: *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*. 2001. v. 3, p. 2523 – 2528. ISSN 1050-4729.

CHU, A.; KAZEROONI, H.; ZOSS, A. On the biomimetic design of the berkeley lower extremity exoskeleton (bleex). In: *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. 2005. p. 4345 – 4352.

CORBET, J. et al. *Linux Device Drivers*. third. : O?Reilly, 2005.

DIFTLER, M. et al. Robonaut 2 - the first humanoid robot in space. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. 2011. p. 2178 – 2183. ISSN 1050-4729.

DOLENC, M.; MARKOVIC, T. *PCI IP Core Design document*. Janeiro 2002.

ETG. Ethercat slave implementation guide. Maio 2009.

ETG. Ethercat the ethernet fieldbus. *Internal Report*, s.d.

FJELDBO, S. J. *Admnistration of remote computer netwoks*. Dissertação (Mestrado) — Oslo University College - Department of Computer Science, May 2005.

FORNER-CORDERO, A. et al. Upper limb exoskeleton for motor control. In: *Proceedings of the 21st International Congress of Mechanical Engineering*. 2011.

FREESCALE. *K40 Sub-Family Data Sheet*. May. 2011.

FSF. *Gnu Lesser General Public License Version 3*. Jun. 2007.

FUCHS, M. et al. Rollin' justin - design considerations and realization of a mobile platform for a humanoid upper body. In: *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*. 2009. p. 4131 – 4137. ISSN 1050-4729.

FUJITA, M.; KAGEYAMA, K. An open architecture for robot entertainment. In: *Proceedings of the first international conference on Autonomous agents*. New York, NY, USA: ACM, 1997. (AGENTS '97), p. 435 – 442. ISBN 0-89791-877-0.

GINGERICH, K. *Wired-Logic Signaling With M-LVDS*. Oct 2002.

GREBENSTEIN, M. et al. The dlr hand arm system. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. 2011. p. 3175 –3182. ISSN 1050-4729.

HEATH, S. *Embedded Systemm Design*. Second. 2003.

HIRUKAWA, H. et al. Humanoid robotics platforms developed in hrp. *Robotics and Autonomous Systems*, v. 48, n. 4, p. 165 – 175, 2004. ISSN 0921-8890.

HIRZINGER, G. et al. Dlr's torque-controlled light weight robot iii-are we reaching the technological limits now? In: *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*. 2002. v. 2, p. 1710 – 1716.

HONDA. Asimo: Frequently asked questions. *Internal Document*, s.d.

IEEE. Ieee standard for a high performance serial bus. *IEEE Std 1394-1995*, 1996.

JARDIM, B. *Atuadores Elásticos em Série Aplicado no Desenvolvimento de um Exoesqueleto Para Membros Inferiores*. Dissertação (Mestrado) — Escola de Engenharia de São Carlos da Unversidade de São Paulo, 2009.

JARDIM, B.; SIQUEIRA, A. A. Development of series elastic actuators for impedance control of an active ankle foot orthosis. In: *Proceedings of COBEM 2009*. 2009.

KAJITA, S.; ESPIAUR, B. Springer handbook of robotics. In: ____. : Springer, 2008. cap. 16. Legged Robots, p. 361 – 389.

KANEHIRO, F. et al. Distributed control system of humanoid robots based on real-time ethernet. In: *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. 2006. p. 2471 –2477.

KANEKO, K. et al. Humanoid robot hrp-3. In: *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. 2008. p. 2471 –2478.

KANEKO, K. et al. Design of prototype humanoid robotics platform for hrp. In: *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*. 2002. v. 3, p. 2431 – 2436 vol.3.

KANEKO, K. et al. Humanoid robot hrp-2. In: *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*. 2004. v. 2, p. 1083 – 1090. ISSN 1050-4729.

KANEKO, K. et al. Hardware improvement of cybernetic human hrp-4c for entertainment use. In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. 2011. p. 4392 –4399. ISSN 2153-0858.

KANEKO, K. et al. Humanoid robot hrp-4 - humanoid robotics platform with lightweight and slim body -. In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. 2011. p. 4400 – 4407. ISSN 2153-0858.

KANEKO, K. et al. Cybernetic human hrp-4c. In: *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*. 2009. p. 7 –14.

KAZANZIDES, P.; THIENPHRAPA, P. Centralized processing and distributed i/o for robot control. In: *Technologies for Practical Robot Applications, 2008. TePRA 2008. IEEE International Conference on*. 2008. p. 84 – 88.

KAZEROONI, H. et al. On the control of the berkeley lower extremity exoskeleton (bleex). In: *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. 2005. p. 4353 – 4360.

KIM, S.; KAZEROONI, H. High speed ring-based distribuited networked control system for real-time multivariables applications. *Proceedings of IMECE2004*, Nov 2004.

KONYEV, M. et al. Low-level control system of a new biped robot ?rotto? *Proceedings of the Twelfth International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, p. 559 – 566, Sept. 2009.

KWA, H. K. et al. Development of the ihmc mobility assist exoskeleton. In: *Robotics and Automation, 2009 IEEE International Conference on*. 2009. p. 2556 – 2562.

LEE, S.; SANKAI, Y. Power assist control for leg with hal-3 based on virtual torque and impedance adjustment. In: *Systems, Man and Cybernetics, 2002 IEEE International Conference on*. 2002. v. 4. ISSN 1062-922X.

LEE, S.; SANKAI, Y. Power assist control for walking aid with hal-3 based on emg and impedance adjustment around knee joint. In: *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*. 2002. v. 2, p. 1499 – 1504.

LIU, H. et al. A dexterous humanoid five-fingered robotic hand. In: *Robot and Human Interactive Communication, 2008. RO-MAN 2008. The 17th IEEE International Symposium on*. 2008. p. 371 – 376.

LOHMEIER, S. *Design and Realization of a Humanoid Robot for Fast and Autonomous Bipedal Locomotion*. Tese (Doutorado) — Technische Universitat Munchen, June 2010.

LOHMEIER, S.; BUSCHMANN, T.; ULBRICH, H. Humanoid robot lola. In: *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*. 2009. p. 775 – 780. ISSN 1050-4729.

MAKINSON, B. Research and development prototype for machine augmentation of human strength and endurance. hardiman i project. *General Electric*, Maio 1971.

MAXIM. *MAX14840E/MAX14841E Datasheet*. Feb. 2010. Rev 0.

MELNYKOV, A. et al. Biped robot "rotto": stiff and compliant. In: *Robotics in Alpe-Adria-Danube Region (RAAD), 2010 IEEE 19th International Workshop on*. 2010. p. 261 – 266.

MICROCHIP. *PIC32MX3XX/4XX Data Sheet*. 2010.

OGURA, Y. et al. Development of a new humanoid robot wabian-2. In: *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. 2006. p. 76 – 81. ISSN 1050-4729.

OH, J.-H. et al. Design of android type humanoid robot albert hubo. In: *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. 2006.

PRATT, J.; KRUPP, B. Design of a bipedal walking robot. In: *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*. 2008. v. 6962.

PRETE, P. *Linux driver for Opencores' PCI BRIDGE*. s.d. Website. Acessado em 20 de Novembro de 2011 - http://sourceforge.net/projects/pcibridgedriver/.

PRYTZ, G. A performance analysis of ethercat and profinet irt. *Proceedings of the 13th IEEE Internacional Conference on Emerging Technologies and Factory Automation (ETFA)*, p. 408–415, 2008.

RATHA, B. Network: Types and topologies. *School of Library and Information Science - Devi Ahilya University, Indore*, s.d.

SAKAGAMI, Y. et al. The intelligent asimo: system overview and integration. In: *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*. 2002. v. 3, p. 2478 – 2483.

SANKAI, Y. Leading edge of cybernics: Robot suit hal. *SICE-ICASE International Joint Conference 2006*, Outubro 2006.

SARKER, M. et al. An ieee-1394 based real-time robot control system for efficient controlling of humanoids. In: *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. 2006.

SAWYER, N. *XAP225 - Data to Clock Phase Alignment*. 2009.

STEGER, R.; KIM, S. H.; KAZEROONI, H. Control scheme and networked control architecture for the berkeley lower extremity exoskeleton (bleex). In: *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. 2006. p. 3469 –3476. ISSN 1050-4729.

SULLIVAN, M. *Secure remote network administration and power management*. Dissertação (Mestrado) — Naval Postgraduate School - Monterey, California, Junho 2004.

THIENPHRAPA, P.; KAZANZIDES, P. Design of a scalable real-time robot controller and application to a dexterous manipulator. In: *in IEEE Int. Conf. on Robotics and Biomimetics (RoBio)*. 2011.

TI. M-lvds signaling rate versus distance. *Texas Intruments Application Report - SLLA127*, Abril 2003.

TI. *SN65MLVD200A , SN65MLVD202A, SN65MLVD204A , SN65MLVD205A - Datasheet*. Dec. 2003.

VAROTO, R. *Desenvolvimento de avaliação de um protótipo de sistema híbrido para membro superior de tetraplégicos*. Dissertação (Master Thesis) — Escola de Engenharia de São Carlos da Universidade de São Paulo, 2010.

VAROTO, R.; BARBARINI, E. S.; CLIQUET, A. A hybrid system for upper limb movement restoration in quadriplegics. *Artificial Organs*, v. 32, n. 9, p. 725 – 729, 2008.

VENKATESWARAN, S. *Essential Linux Device Drivers*. first. : Prentice Hall, 2008.

VUKOBRATOVIC, M.; HRISTIC, D.; STOJILJKOVIC, Z. Development of active anthropomorphic exoskeletons. *Medical and Biological Engineering and Computing*, v. 12, n. 1, p. 66–80, Janeiro 1974.

WAIN, R. et al. *An overview of FPGAs and FPGA programming: Initial experiences at Daresbur*. Novembro 2006.

XILINX. *Spartan-3AN FPGA Family Data Sheet*. Apr. 2011.

YOKOI, K. et al. Experimental study of humanoid robot hrp-1s. *The International Journal of Robotics Research*, p. 351 – 362, May. 2004.

YU, Z. et al. Distributed control system for a humanoid robot. In: *Mechatronics and Automation, 2007. ICMA 2007. International Conference on*. 2007. p. 1166 – 1171.

ZOSS, A.; KAZEROONI, H.; CHU, A. On the mechanical design of the berkeley lower extremity exoskeleton (bleex). *Proc. of IEEE Intelligent Robots and Systems*, p. 2053–2058, Agosto 2005.

ZOSS, H. K. A.; CHU, A. Biomechanical design of the berkeley lower extremity exoskeleton (bleex). *IEEE/ASME Tansactions on Mechatronics*, v. 11, n. 2, p. 128–137, Abril 2006.

APÊNDICE A – REGISTRADORES DO CONTROLADOR DE REDE

Tabela 2: Registradores de Controle

NOME	ENDEREÇO
READ_COMMAND	0x00
INTERRUPT_STATUS	0x04
TX_BASE_ADDRESS	0x08
RX_BASE_ADDRESS	0x0c
RING_BUFFER_STATUS	0x10
CHANNEL_ENABLE	0x14
CHANNEL_RESET	0x18
MASTER_RESET	0x1c
CHANNEL_REMAP	0x20

Tabela 3: Descrição do registrador READ_COMMAND

READ_COMMAND		
BIT #	**NOME**	**DESCRIÇÃO**
0 - 29	READ SIZE	Quantidade de dados à serem lidos expressa em DWORDs.
30 - 31	BUFFER NUMBER	Número do buffer a ser lido.

Tabela 4: Descrição do registrador INTERRUPT_STATUS

INTERRUPT_STATUS		
BIT #	**NOME**	**DESCRIÇÃO**
0	TX_B1_COMPLETE	Leitura dos dados do TxBuffer 1 completa.
1	TX_B2_COMPLETE	Leitura dos dados do TxBuffer 2 completa.
2	RX_B1_COMPLETE	Dados escritos no RxBuffer 1.
3	RX_B2_COMPLETE	Dados escritos no RxBuffer 2.
4	RX_B3_COMPLETE	Dados escritos no RxBuffer 3.
5	RX_B4_COMPLETE	Dados escritos no RxBuffer 4.
6	RX_B5_COMPLETE	Dados escritos no RxBuffer 5.
7	RX_B6_COMPLETE	Dados escritos no RxBuffer 6.
8	RX_B7_COMPLETE	Dados escritos no RxBuffer 7.
9	RX_B8_COMPLETE	Dados escritos no RxBuffer 8.
10	RX_B9_COMPLETE	Dados escritos no RxBuffer 9.
11	RX_B10_COMPLETE	Dados escritos no RxBuffer 10.
12	CH0_ERROR	Erro de envio no canal 0.
13	CH1_ERROR	Erro de envio no canal 1.
14	CH2_ERROR	Erro de envio no canal 2.
15	CH3_ERROR	Erro de envio no canal 3.
16	CH4_ERROR	Erro de envio no canal 4.
17	CH5_ERROR	Erro de envio no canal 5.
18	CH6_ERROR	Erro de envio no canal 6.
19	CH7_ERROR	Erro de envio no canal 7.
20 - 31	-	Não implementado.

Tabela 5: Descrição do registrador TX_BASE_ADDRESS

TX_BASE_ADDRESS		
BIT #	**NOME**	**DESCRIÇÃO**
0 - 30	BASE_ADDR	30 Bits menos significativos do endereço do início da memória em que se encontram os TxBuffers.
31	-	Não implementado.

Tabela 6: Descrição do registrador RX_BASE_ADDRESS

RX_BASE_ADDRESS		
BIT #	**NOME**	**DESCRIÇÃO**
0 - 30	BASE_ADDR	30 Bits menos significativos do endereço do início da memória em que se encontram os RxBuffers.
31	-	Não implementado.

Tabela 7: Descrição do registrador RING_BUFFER_STATUS

RING_BUFFER_STATUS		
BIT #	**NOME**	**DESCRIÇÃO**
0 - 9	BUFF_(n+1)_BUSY	Para cada bit **n**: No caso de leitura do registrador, representa que o RxBuffer **n+1** não se encontra disponível para escrita de dados. No caso de escrita para o registrador, libera o RxBuffer **n+1**.
10 - 31	-	Não implementado.

Tabela 8: Descrição do registrador CHANNEL_ENABLE

CHANNEL_ENABLE		
BIT #	**NOME**	**DESCRIÇÃO**
0 - 7	CH_n_EN	Para cada bit **n**: Habilita o Canal **n**.
8 - 31	-	Não implementado.

Tabela 9: Descrição do registrador CHANNEL_RESET

CHANNEL_RESET		
BIT #	**NOME**	**DESCRIÇÃO**
0 - 7	CH_n_RST	Para cada bit **n**: Reinicia o Canal **n**.
8 - 31	-	Não implementado.

Tabela 10: Descrição do registrador MASTER_RESET

MASTER_RESET		
BIT #	**NOME**	**DESCRIÇÃO**
0	RESET	Reinicia o Controlador de Rede.
1 - 31	-	Não implementado.

Tabela 11: Descrição do registrador CHANNEL_REMAP

CHANNEL_REMAP		
BIT #	**NOME**	**DESCRIÇÃO**
0 - 7	CH_n_REMAP	Para cada bit **n**: Força o Canal **n** a remapear os endereços.
8 - 31	-	Não implementado.

Tabela 12: Registradores de Status dos Canais

NOME	ENDEREÇO
CH0_SLV_MAP	0x24
CH0_CUR_MAP	0x28
CH0_ERROR_DESC	0x2c
CH1_SLV_MAP	0x30
CH1_CUR_MAP	0x34
CH1_ERROR_DESC	0x38
CH2_SLV_MAP	0x3c
CH2_CUR_MAP	0x40
CH2_ERROR_DESC	0x44
CH3_SLV_MAP	0x48
CH3_CUR_MAP	0x4c
CH3_ERROR_DESC	0x50
CH4_SLV_MAP	0x54
CH4_CUR_MAP	0x58
CH4_ERROR_DESC	0x5c
CH5_SLV_MAP	0x60
CH5_CUR_MAP	0x64
CH5_ERROR_DESC	0x68
CH6_SLV_MAP	0x6c
CH6_CUR_MAP	0x70
CH6_ERROR_DESC	0x74
CH7_SLV_MAP	0x78
CH7_CUR_MAP	0x7c
CH7_ERROR_DESC	0x80

Tabela 13: Descrição do registrador CHx_SLV_MAP

CHx_SLV_MAP		
BIT #	**NOME**	**DESCRIÇÃO**
0 - 1	-	Não implementado.
2 - 31	ADDR_n_MAPPED	Cada bit **n** representa se o endereço **n** se encontra mapeado no canal x. Ex: Se o Bit 2 estiver com nível logico 1 representa que o endereço n°2 foi mapeado.

Tabela 14: Descrição do registrador CHx_CUR_MAP

CHx_CUR_MAP		
BIT #	**NOME**	**DESCRIÇÃO**
0 - 1	-	Não implementado.
2 - 31	ADDR_n_MAPPED	Cada bit **n** representa se o endereço **n** respondeu no último ciclo da Linha Poll no Canal x. Ex: Se o Bit 2 estiver com nível logico 1 representa que o endereço n°2 do Canal x respondeu quando enviado um comando POLL para o seu endereço no último ciclo.

Tabela 15: Descrição do registrador CHx_ERROR_DESC

CHx_ERROR_DESC		
BIT #	**NOME**	**DESCRIÇÃO**
0 - 7	ERROR_ADDR	Endereço do destinatário da mensagem relativa ao último erro de transmissão ocorrido no Canal x. Caso nenhum erro tenha ocorrido até então, este registrador não tem significado
8 - 15	ERROR_CMD	Comando da mensagem relativa ao último erro de transmissão ocorrido no Canal x. Caso nenhum erro tenha ocorrido até então, este registrador não tem significado
16 - 23	ERROR_ID	ID da mensagem relativa ao último erro de transmissão ocorrido no Canal x. Caso nenhum erro tenha ocorrido até então, este registrador não tem significado
24 - 31	ERROR_SZ	Tamanho do campo de dados da mensagem relativa ao último erro de transmissão ocorrido no Canal x. Caso nenhum erro tenha ocorrido até então, este registrador não tem significado

APÊNDICE B – FUNÇÕES DISPONÍVEIS PELA API E PELO MANAGER

rb_desc_t* rb_init(void)

Descrição:	Aloca **r_bone** na memória, abre driver, mapeia **tx_buffer** e **rx_buffer** e retorna endereço de **r_bone**.
Arquivo:	rb_api.c
Include:	rb_api.h

uint32_t rb_read(uint32_t *value, uint32_t addr, int resource)

Descrição:	Realiza leitura no Controlador de Rede do endereço **addr** e grava na variável cujo ponteiro **value** aponta. Retorna **-1** em caso de erro e **0** em caso de sucesso na leitura. Para **resource = 0** o mapa de memória é relacionado ao controlador PCI, para **resource = 1** o mapa de memória é relacionado à Interface Wishbone.
Arquivo:	rb_api.c
Include:	rb_api.h

uint32_t rb_write(uint32_t value, uint32_t addr, int resource)

Descrição:	Realiza escrita da variável **value** no endereço **addr** do Controlador de Rede. Retorna **-1** em caso de erro e **0** em caso de sucesso na escrita. Para **resource = 0** o mapa de memória é relacionado ao controlador PCI, para **resource = 1** o mapa de memória é relacionado à Interface Wishbone.
Arquivo:	rb_api.c
Include:	rb_api.h

uint32_t rb_seek(uint32_t offset)

Descrição:	Modifica endereço de leitura e de escrita para o endereço **offset**.
Arquivo:	rb_api.c
Include:	rb_api.h

uint32_t rb_command_read(uint32_t buffer_n, uint32_t size)

Descrição:	Realiza escrita do valor **size** relacionado ao TxBuffer **buffer_n** no registrador COMMAND_READ do Controlador de Rede.
Arquivo:	rb_api.c
Include:	rb_api.h

uint32_t uint32_t rb_check_buffer(void)

Descrição:	Retorna se existe pendencia de leitura em algum RxBuffer
Arquivo:	rb_api.c
Include:	rb_api.h

uint32_t rb_close(void)

Descrição:	Libera memória utilizada por r_bone e fecha driver.
Arquivo:	rb_api.c
Include:	rb_api.h

node_t *rb_mk_node(uint8_t ch, uint8_t addr,
node_reg_t r_reg_list[],
node_reg_t w_reg_list[])

Descrição:	Cria imagem na memória de um Nó do Canal **ch**, endereço **addr**, *read registers* **r_reg_list** e *write registers* **w_reg_list**. Retorna endereço da imagem alocada.
Arquivo:	rb_node.c
Include:	rb_node.h

uint32_t rb_make_buffer(node_list_t node_list[],
uint32_t tx_buffer[],
request_desc_t request_table[])

Descrição:	Escreve mensagens no **tx_buffer** referentes às solicitações escritas na lista **node_list** e aloca ID da **request_table** quando necessário. Retorna quantidade de DWORDs escritas no buffer.
Arquivo:	rb_node.c
Include:	rb_node.h

void rb_parse_buffer(uint32_t rx_buffer[],
request_desc_t request_table[],
uint32_t buffer_offset)

Descrição:	Interpreta mensagens recebidas no **rx_buffer**, utilizando o offset **buffer_offset** e atualiza a lista de IDs **request_table**.
Arquivo:	rb_node.c
Include:	rb_node.h

void clr_id_tbl(request_desc_t requesta_table[])

Descrição:	Desaloca todos os IDs da tabela **request_table**.
Arquivo:	rb_mngr.c
Include:	rb_mngr.h

MNGR_RET mngr_chk_req(rb_mngr_t *mngr)

Descrição:	Verifica se existem solicitações pendentes.
Arquivo:	rb_mngr.c
Include:	rb_mngr.h

MNGR_RET mngr_wt_buff(rb_mngr_t *mngr)

Descrição:	Escreve mensagens para rede de acordo com dados passado para o **mngr**.
Arquivo:	rb_mngr.c
Include:	rb_mngr.h

void mngr_init(rb_mngr_t *mngr)

Descrição:	Inicializa o sistema.
Arquivo:	rb_mngr.c
Include:	rb_mngr.h

uint32_t mngr_buff_parse(rb_mngr_t *mngr)

Descrição:	Interpreta todas mensagens recebidas em todos os RxBuffers.
Arquivo:	rb_mngr.c
Include:	rb_mngr.h