UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

**ERNESTO DORNELLES PINTO**

# PROJETO DE DIPLOMAÇÃO

# APLICAÇÃO DE PLATAFORMA ARM7 PARA NÓ EM REDES CAN

Porto Alegre
2010

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

# APLICAÇÃO DE PLATAFORMA ARM7 PARA NÓ EM REDES CAN

Projeto de Diplomação apresentado ao Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul, como parte dos requisitos para Graduação em Engenharia Elétrica.

ORIENTADOR: Prof. Dr. Valner João Brusamarello

Porto Alegre
2010

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

ERNESTO DORNELLES PINTO

# APLICAÇÃO DE PLATAFORMA ARM7 PARA NÓ EM REDES CAN

Este projeto foi julgado adequado para fazer jus aos créditos da Disciplina de "Projeto de Diplomação", do Departamento de Engenharia Elétrica e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____

Prof. Dr. Valner João Brusamarello, UFRGS

Doutor pela UFSC – Florianópolis, Brasil

Banca Examinadora:

Prof. Dr. Alexandre Balbinot, UFRGS

Doutor pela UFRGS – Porto Alegre, Brasil

Prof. Dr. Marcelo Soares Lubaszewski, UFRGS

Doutor pelo Institut National Polytechnique de Grenoble – Grenoble, França.

Prof. Dr. Valner João Brusamarello, UFRGS

Doutor pela UFSC – Florianópolis, Brasil

Porto Alegre, dezembro de 2010

# DEDICATÓRIA

Dedico este trabalho aos meus pais e irmãos, que em todos os momentos estão dispostos a me ajudar em especial pela dedicação e apoio em todos os momentos difíceis.

# AGRADECIMENTOS

Ao Prof. Dr. Valner J. Brusamarello pela orientação no projeto de diplomação.

À UFRGS que oferece recursos necessários para o aperfeiçoamento dos recursos humanos.

Ao Engenheiro Angelo Susin que disponibilizou o *encoder* e a placa McBoard LPC2368 para o desenvolvimento dos trabalhos.

Ao meu irmão Américo, pela ótima pessoa que é e pela amizade que construímos.

Aos meus pais, Américo Jardim Pinto e Maria Glaci, pela dedicação e afeto.

À minha família.

Aos meus amigos pelas horas de descontração e apoio.

# RESUMO

Os sistemas de automação e controle embarcados estão presentes, atualmente, em áreas que vão desde aplicações móveis até equipamentos industriais. Em sistemas de arquitetura distribuída, o protocolo de comunicação que interliga todos os pontos da rede desempenha um papel de fundamental importância. O Protocolo digital serial CAN – Controller Area Network – é empregado em aplicações diversas de forma a garantir critérios de segurança e desempenho. Neste trabalho, é utilizada uma plataforma ARM7 para realizar o nó de uma rede CAN. Foi desenvolvido o *firmware* para integração do dispositivo em uma rede CANopen e o mesmo foi validado por meio do estabelecimento da comunicação com um transdutor industrial contendo uma interface de mesmo protocolo.

**Palavras-chaves: Engenharia Elétrica. Redes de comunicação CAN. Automação e Controle. Eletrônica e Instrumentação.**

# ABSTRACT

The automation and control embedded systems are present today, in areas ranging from mobile applications to industrial equipment. In distributed systems architecture, the communication protocols that connect all these systems makes an important role. The digital serial protocol CAN – Controller Area Network – is used in various applications to ensure safety and performance criteria. In this work, is used an ARM7 platform to perform the node for a CAN network. It developed the firmware for device integration in a CANopen network and it was validated by establishing communication with a transducer containing an industrial interface same protocol.

**Keywords: Electrical Engineering. CAN Communication networks. Automation and Control. Electronic and Instrumentation.**

# SUMÁRIO

# LISTA DE ILUSTRAÇÕES

# LISTA DE TABELAS

# LISTA DE ABREVIATURAS

ECU: *Electronic Control Unit*

CAN: *Controller Area Network*

DLC: *Data Length Code*

RTR: *Remote Transmission Request*

MCU: Microcontrolador

RID: *Receive Identifier Register*

RDA: *Receive Dara Register* A

RDB: *Receive Dara Register* B

TID: *Transmit Identifier Register*

TDA: *Transmit Dara Register* A

TDB: *Transmit Dara Register* B

COB: *Communication Objects*

PDO: *Process Data Objects*

SDO: *Service Data Objects*

NMT: *Network Manage*

# 1. INTRODUÇÃO

Um sistema embarcado é a combinação de um conjunto de dispositivos eletro-eletrônicos microprocessado e software desenvolvido para executar uma ou mais funções específicas, como controle, monitoramento e comunicação sem a intervenção humana [1-2]. Ao longo dos anos, a indústria automotiva lançou mão de sistemas embarcados para o controle das muitas funções presentes em automóveis. Contudo, as tecnologias em poucos anos migraram para outras áreas, como caminhões, barcos, máquinas agrícolas, de construções civis e militares [3].

Atualmente, observa-se que os sistemas de controle são desenvolvidos de modo que cada um seja responsável por uma determinada função no veículo. No entanto, em sistemas interligados, no qual cada um é responsável por uma parte do veículo e todos compartilham informações, o controle sobre os vários dados eletrônicos é mais fácil de ser atingido [4]. Seguindo esta filosofia de desenvolvimento, os sistemas empregados pelas montadoras e fornecedores de componentes automotivos possibilitam a percepção de que o controle do veículo é integrado, isto é, de que há uma unidade central inteligente.

Os sistemas de controle, no setor automotivo, podem ser desenvolvidos e interconectados de duas formas distintas, atualmente em destaque, em uma arquitetura concentrada ou distribuída.

Na arquitetura centralizada, há uma única unidade eletrônica de controle (ECU – *electronic control unit*), a qual é responsável por todo o sistema. Esse tipo de arquitetura tem as vantagens de possuir um hardware relativamente simples e todos os dados de entrada disponíveis durante toda a operação do sistema. Porém, a possível necessidade de uma numerosa quantidade de cabos para a conexão de sensores e atuadores à ECU e a limitação da expansão do sistema são alguns dos fatores de desvantagens no uso dessa arquitetura.

Na arquitetura distribuída, as diversas funções existentes no veículo são divididas por várias ECUs interligadas. Entre as vantagens dessa arquitetura, pode ser destacado o número reduzido de cabos, uma vez que as ECUs estão próximas dos sensores e atuadores, a maior robustez do sistema de controle, a possibilidade de ampliação do mesmo – impactando num mínimo de alterações no sistema de controle do veículo – e a modularização do projeto, facilitando a implantação de testes de validação [5]. No entanto, uma arquitetura distribuída exige um protocolo de comunicação entre as ECUs – nós da rede, e consequentemente, um software de controle para a rede de comunicação.

A decisão do tipo de arquitetura mais apropriada varia com a aplicação, isto é, depende principalmente da complexidade do sistema a ser controlado (número de variáveis de entrada e saída e tamanho físico do sistema), dos componentes eletrônicos necessários às ECUs, sensores e atuadores, da robustez e custo total. A análise desses fatores é um trabalho de engenharia de produto, envolvendo as perspectivas de tecnologia, marketing e de orçamento das empresas.

Do ponto de vista do avanço tecnológico e da possibilidade de possíveis expansões, a arquitetura distribuída desponta como melhor opção [4-5]. Esta arquitetura caracteriza-se como uma rede multiplexada – *networking*, na qual as ECUs transferem os dados através de um meio serial. O protocolo de comunicação da rede precisa garantir a interoperabilidade entre os equipamentos de diversos fabricantes, pois o funcionamento interno destes é sigiloso, a rigor. Uma vez que protocolos seriais são tecnicamente mais adequados, a escolha pelo protocolo CAN – *controller area network* – é indicada por este ser um destaque dentro dessa classe [1], [6]. Este protocolo foi desenvolvido na década de oitenta por Robert Bosh GmbH para uso em unidades de controle eletrônico em automóveis. Entre outros fatores, a popularidade das aplicações com CAN devem-se às suas características de configuração da taxa de transmissão, aos mecanismos de identificação e tratamento de erros e a flexibilidade

de mudanças de dispositivos, que favorecem operações de manutenção e alterações no sistema.

A necessidade de engenheiros da Bosch por uma rede que integrasse os sistemas de controle em veículos automotivos de passeio viabilizou o desenvolvimento, e posteriormente a integração do CAN na indústria. Atualmente, este protocolo de rede está entre aqueles de maior sucesso, sendo que quase todos automóveis fabricados na Europa são equipados com ao menos uma rede CAN [1]. As vantagens deste protocolo, comparada a outras soluções de rede, estão baseadas em sua robustez e razão custo/benefício [6].

O CAN é usado em uma variedade de aplicações relacionadas a qualquer parte da vida humana. Cita-se [7]:

- Transporte: originalmente usado em veículos de passeio, hoje esta presente em veículos pesados, marítimos, trens e outros;
- Manufatura: usado numa ampla diversidade de indústrias, principalmente no controle embarcado de máquinas, mas também para a automação de plantas, processos industriais e geração de potência;
- Construção: usado na rede embarcada de máquinas de construção. Ele é também usado na automação de construções, por exemplo, no controle de elevadores, controle embarcado de portas e HVAC – *Heating, Ventilation and Air-conditioning*;
- Agricultura: usado em equipamentos agrícolas estacionários bem como em veículos florestais e agrícolas;
- Saúde: usado na rede de controle embarcado para dispositivos médicos e unidades de terapia intensiva;

- Comunicação: usado para o envio, empacotamento e controle embarcado de equipamentos de telecomunicações. Por exemplo, máquinas de triagem e transportadoras de cartas estão usando o CAN para controle embarcado;
- Varejo e Finanças: usado como controle embarcado em máquinas de venda incluindo equipamentos de postos de combustíveis e caixas eletrônicos;
- Entretenimento: usado em sistemas de controle de estúdios e palcos, por exemplo, controle de iluminação e portas, máquinas de jogos de azar e brinquedos;
- Ciência: usado como uma rede de controle embarcada, por exemplo, em experimentos de física de alta energia (aceleradores de prótons) e telescópios astronômicos.

A indústria de veículos desenvolve inovações onde mais de 80% são provenientes de sistemas eletrônicos, tais como: controle de bloqueio de portas, sistema antitravamento das rodas – *ABS*, sensores de pressão dos pneus, suspensão ativa, sistema de posicionamento global – *GPS*, *Airbags*, piloto automático, entre outros [8], [13].

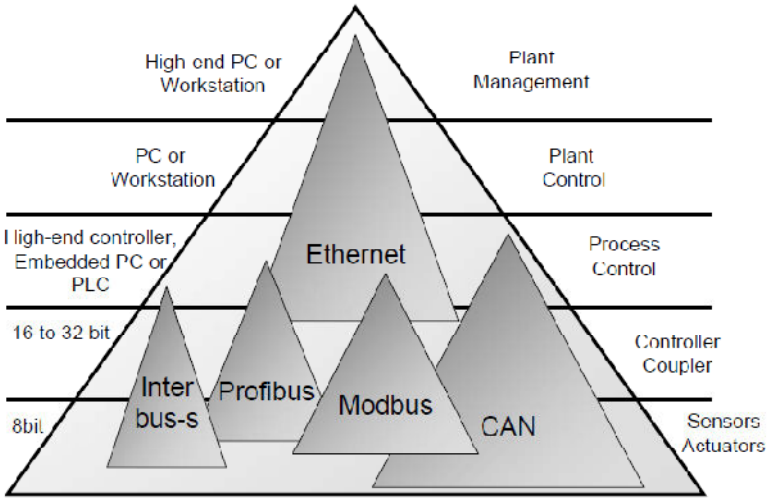


**Figura 1.1: Distribuição da comunicação de sistemas na pirâmide da automação [9].**

Hoje o CAN é uma das melhores escolhas para aplicações de redes embarcadas que precisam comunicação entre vários microcontroladores de 8-*bit*, 16-*bit* e 32-*bit*, conforme mostrado na Figura 1.1 [9]. Existem no mercado cerca de 20 diferentes fabricantes de microcontroladores com interfaces CAN, tais como: Infineon, Intel, Microchip, Motorola e Philips.

No contexto apresentado, as redes de comunicação de dados com CAN satisfazem a crescente necessidade de alto grau de segurança e comodidade existente em setores que envolvem vidas humanas. Este projeto tem como objetivo principal desenvolver um *firmware* para uma plataforma ARM7 de forma que esta atue como um nó de uma rede CAN. Este nó pode ser considerado como um sistema embarcado de monitoramento de variáveis de controle e a sua comunicação de dados é por meio do protocolo de alto nível CANopen. Os detalhes do hardware e do software utilizados são analisados.

# 2. CONTEXTO DO PROJETO

O trabalho desenvolvido descreve a montagem de um protótipo de nó para uma rede CAN realizado com uma plataforma ARM7. Para alcançar este objetivo, o entendimento do funcionamento da rede faz-se necessário, desta forma neste capítulo é exposta uma descrição das características gerais do protocolo e posteriormente as especificações do projeto.

## 2.1. O Protocolo CAN

O CAN é um protocolo serial e assíncrono. O mesmo foi padronizado internacionalmente e documentado na ISO – *International Standarts Organization* – 11898, para redes de alta velocidade (entre 125Kbps e 1Mbps), e ISO11519, para redes de baixa velocidade (entre 10Kbps e 125Kbps), e é recomendado pela SAE – *Society of Automotive Engineers*. Estes padrões caracterizam a camada física e camada de enlace de dados, respectivamente 1 e 2, conforme o modelo de referência para desenvolvimento de redes computacionais OSI – *Open Systems Interconnection*, ISO7498, conforme mostrado na Fig. 2.1.1 [10].

A camada física define os níveis de tensão elétrica, impedância dos cabos, tempos de *bits*, codificação destes e a sincronização. A camada de dados é definida pelo encapsulamento, desencapsulamento, codificação dos quadros, controle de acesso ao meio, detecção e sinalização de erro. Os protocolos destinados para as camadas de 3 a 7 dependem das necessidades de cada aplicação, habilitando o desenvolvimento de padrões próprios, baseados em protocolos de alto nível – *Higher Layer Protocols*. Entre outros padrões, pode-se destacar o CANopen (para diversas aplicações), NMEA2000 (para aplicações navais e aéreas), o SAEJ1939 (para aplicações automotivas) e o ISO11783 (para aplicações agrícolas).

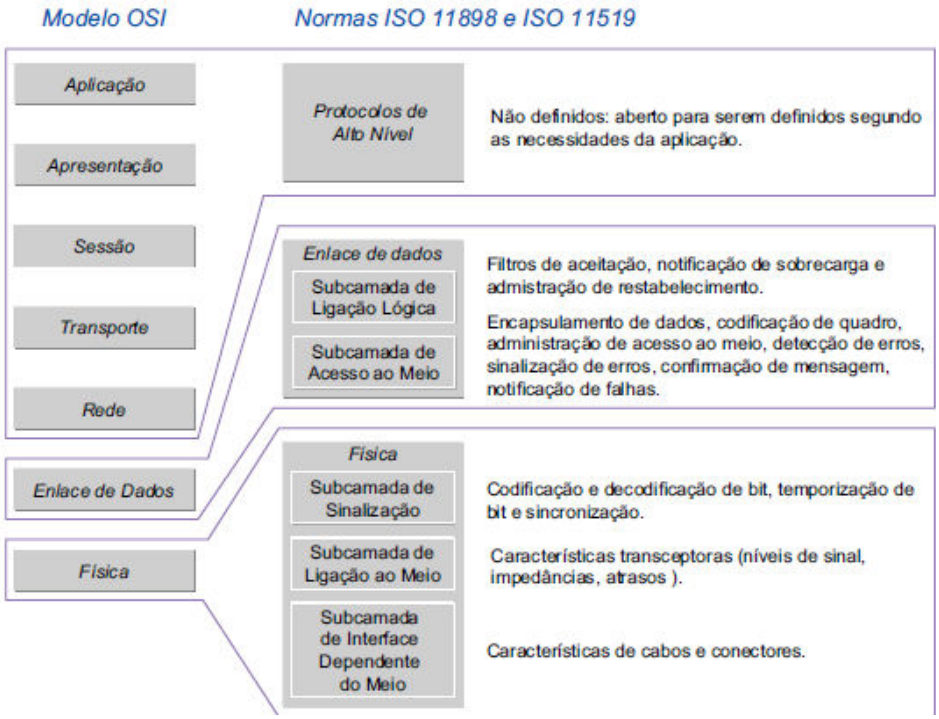


**Figura 2.1.1: Relações entre as camadas do modelo ISO/OSI e padrões ISO11898 e ISO11519 [10].**

A evolução da rede multiplexada em veículos automotivos fez a SAE normalizar os protocolos em três classes A, B e C. Os protocolos de classe A tem velocidade de transmissão inferior a 10Kbps, muitas vezes não exigindo hardware microprocessado, e são utilizados para comunicar sensores e controles que não tem característica crítica, tais como áudio, *displays* e alguns diagnósticos. Os protocolos de classe B tem velocidade de 10Kbps a 125Kbps, e são usados na comunicação entre ECUs. Os protocolos de classe C tem velocidades de 125Kbps a 1Mbps, e são usados para controle em tempo real geralmente de algum sistema de segurança do veículo.

### 2.1.1. Funcionamento do CAN

O enlace entre as ECUs de um sistema numa rede CAN constitui o barramento - *CANbus*, Figura 2.1.1.1. O ato de projetá-lo é de fundamental importância, pois o sincronismo

das operações das ECUs depende do tempo de propagação física das mensagens no barramento. O sincronismo entre os módulos (ECU, sensores e atuadores) conectados à rede é feito a partir do início de cada mensagem entregue ao barramento, cuja ocorrência é em períodos de tempo conhecidos e constantes. A taxa de transmissão de dados é inversamente proporcional ao comprimento do barramento, pois o esquema de ar*bit*ragem necessita que a onda elétrica presente no barramento se propague até o nó mais remoto e volte antes de ser amostrada. Na transmissão de cada *bit*, o sinal tem tempo suficiente de estabilizar-se, de forma praticamente simultânea em todos os nós. Há a possibilidade de configurá-la desde alguns Kb/s até 1 Mb/s, cujo alcance é de 40 metros no máximo. A rede pode ser configurada de forma ponto a ponto (dois dispositivos trocando informações), *broadcast* (de um dispositivo a todos) ou *multicast* (de um dispositivo a um grupo). Outra característica interessante é de duas ou mais sub-redes operarem com velocidades diferentes na mesma aplicação, contudo há módulos – *Gateway* – que interligam as sub-redes.

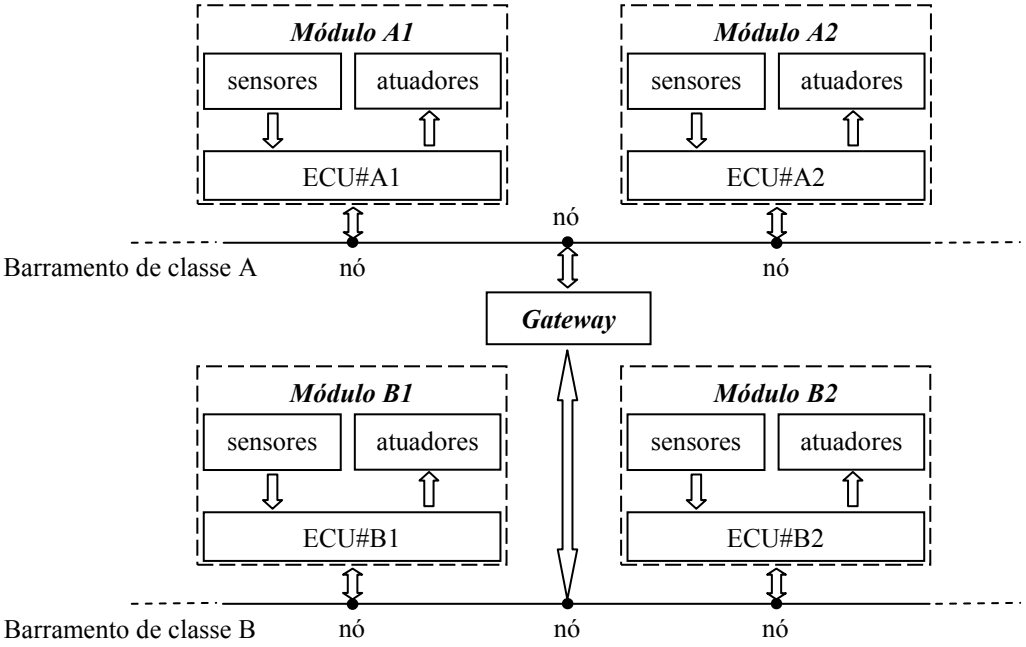


**Figura 2.1.1.1: Topologia de rede de dois barramentos CAN com velocidades diferentes.**

A idéia de um sistema multi-mestre, no qual os diferentes módulos presentes na rede podem figurar entre mestres e escravos em diferentes momentos, é um método de arbitragem para o acesso ao meio de transmissão de dados que evita colisões e permite uma resposta rápida à necessidade de transmissão, denominado CSMA/CD *with NDA – Carrier Sense Multiple Access / Collision Detection with Non-Destructive Arbitration*. Neste método de acesso não-destrutivo, os módulos verificam o estado do barramento, avaliando se há outro de maior prioridade transmitindo mensagens. Em caso afirmativo, o módulo de menor prioridade pára sua transmissão, e o de maior prioridade mantém-se enviando mensagens normalmente. Todos os nós da rede são capazes de ler, porém somente um dispositivo pode escrever no barramento por vez.

### 2.1.2. Mensagens do CAN

A rede de comunicação com o CAN é fundamentada em mensagens formadas por quadros de *bits – frames*, distribuídos em campos que desempenham determinadas funções, designando informações que vão desde o módulo transmissor até o assunto enviado ou recebido. A transferência de mensagens é controlada e manifestada por meio de quatro diferentes tipos de *frames*:

- *Data Frame*: contém os dados de um transmissor para o receptor;
- *Remote Frame*: é transmitido a um ou vários módulos da rede para solicitar a transmissão do *Data Frame* com o mesmo identificador;
- *Error Frame*: é transmitido por qualquer unidade na detecção de erro no barramento;
- *Overload Frame*: é usado para promover um atraso extra entre o *Data Frame* e o *Remote Frame* anterior e seguinte.

Um nó pode requisitar um serviço ou uma informação no barramento de uma ou várias ECUs por meio de um *Remote Frame* – que não apresenta campo de dados, o nó responsável por realizar a leitura identificará a necessidade de disponibilizar tal informação e a transmite – *Data Frame*, obedecendo a sua prioridade. O nó receptor reconhece a mensagem por meio de seu conteúdo, pois não há um endereço explícito nas mensagens. Isto é, há a identificação da variável transmitida e não o endereço onde a mesma está.

Há duas versões do CAN que diferem entre si pelo tamanho do identificador, o CAN 2.0A – *Standard* CAN – de 11 *bits* e o CAN 2.0B – *Extended* CAN – de 29 *bits*.

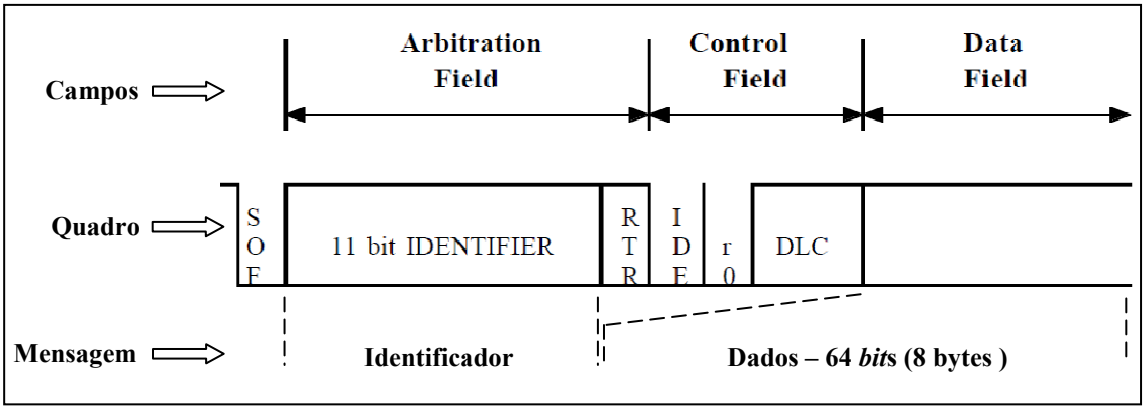


**Figura 2.1.2.1: Formato do *Data Frame* e *Remote Frame* de mensagens CAN2.0A [11].**

Os *frames* mais importantes são o *Data Frame* e o *Remote Frame*. Estes *frames* de mensagens CAN 2.0A, Figura 2.1.2.1 [11], iniciam por um *bit* dominante SOF – *Start Of Frame*. Após um *idle-time*, na descida da borda, ele é usado para sincronização dos diferentes nós da rede. Há o campo de arbitragem, formado pelo identificador de 11 *bits* e pelo *bit* RTR – *Remote Transmission Request*, dominante para *Data Frames* e recessivo para *Remote Frame*. O próximo é o campo de controle composto pelo *bit* IDE – *Identifier Extension Bit* (dominante para *Standard* e recessivo para *Extended*), um *bit* reservado R0 e pelo DLC – *Data Length Code* (conjunto de quatro *bits* que indicam o número de bytes no campo de

dados). A seguir tem-se o campo de dados, onde a informação é transportada, cujo tamanho é de 0 a 64 *bits*. Tanto o campo CRC – *Cyclic Redundancy Check* de 15 *bits* quanto o campo ACK de dois *bits* são responsáveis pela detecção de erros. O fim da mensagem é feita por 7 *bits* recessivos que compõem o campo EOF – *End Of Frame*. No final de cada *frame* são enviados 3 *bits* recessivos – *intermission* para separar *frames* consecutivos.

Os *frames* de mensagens CAN 2.0B diferem dos CAN 2.0A pelo campo de ar*bit*ragem e controle, pois todos os outros campos são idênticos, Figura 2.1.2.2 [11]. Em CAN 2.0B, o campo de ar*bit*ragem é formado pelo identificador base de 11 *bits*, pelo *bit* SRR – *Substitute Remote Request* recessivo de modo a garantir a prioridade do *Standard* na hipótese de dois *frames* de formatos diferentes terem o mesmo identificador base, pelo *bit* IDE, por uma extensão do identificador de 18 *bits* e pelo *bit* RTR. O campo de controle é somente acrescido de um *bit* reservado R1.

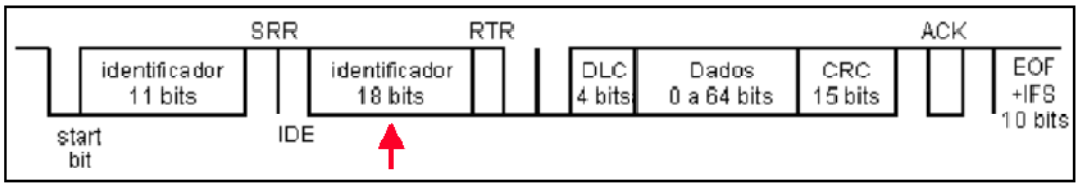


**Figura 2.1.2.2: Formato do *Data Frame* e *Remote Frame* de mensagens CAN2.0B [11].**

O campo identificador (ID) determina a prioridade da mensagem e é exclusivo para cada nó. A hierarquia dos nós é estabelecida a partir do menor valor presente neste campo, possibilitando que os diferentes pontos da rede possam escalonar seu acesso ao barramento. As aplicações com CAN trabalham com interrupções em *real-time* e as mensagens entregues no barramento são acessíveis a qualquer nó, porém estes podem utilizar filtros de aceitação que restringem o processamento da ECU, apenas quando há mensagens de interesse a ela. A quantidade de filtros é função da disposição do circuito controlador de protocolo. Na

realização da rede, a parte mais voltada à aplicação é o dicionário de dados – *data dictionary* o qual descreve as mensagens existentes na aplicação, identificadores e dados, e as ECUs responsáveis pela sua transmissão e recepção.

### 2.1.3. Aspectos físicos do CAN

Os elementos básicos de um padrão CAN, atualmente [11], são o transceptor – *transceiver*, barramento e microcontrolador. O *transceiver* é responsável pela adaptação dos níveis de tensão, de impedâncias e proteção do nó a faltas. O microcontrolador – MCU reúne um módulo de execução e controle do CAN – controlador CAN, além de ter CPU, memória, interfaces de I/O – *input/output* – e demais requisitos para o desenvolvimento de protocolos de mais alto nível. O barramento é formado por 1, 2 ou 4 fios e por terminações. As redes de 2 e 4 fios tem dois sinais de dados, CAN_H – CAN *High* e CAN_L – CAN *Low*, e no caso de 4 fios, possuem também a alimentação – VCC e a referência – GND. Em ambas, as informações são baseadas na diferença de potencial elétrico entre CAN_H e CAN_L, constituindo um par diferencial, o que atenua os efeitos de interferências eletromagnéticas. Os dados são representados por *bits* dominantes e recessivos, conforme o estado de CAN_H e CAN_L, Figura 2.1.3.1.

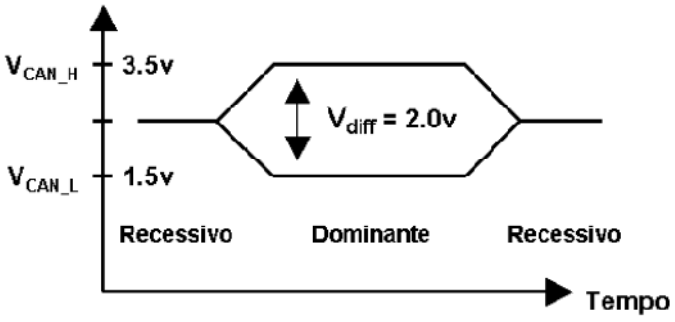


**Figura 2.1.3.1: Níveis de tensão em uma rede CAN.**

Cada *bit* é transmitido por um nível de tensão elétrica específica e constante, conceito denominado NRZ – *Non Return to Zero*. Um *bit* recessivo é representado por CAN_H e CAN_L num nível de tensão elétrica de 2,5V, e um dominante por CAN_H em 3,5V e CAN_L em 1,5V.

As terminações são resistores de 120ohms colocados em CAN_H e CAN_L das ECUs conectadas nos extremos da rede, conforme a Figura 2.1.3.2. Nesta pode ser visto também as medidas que devem ser seguidas para o correto funcionamento do barramento.

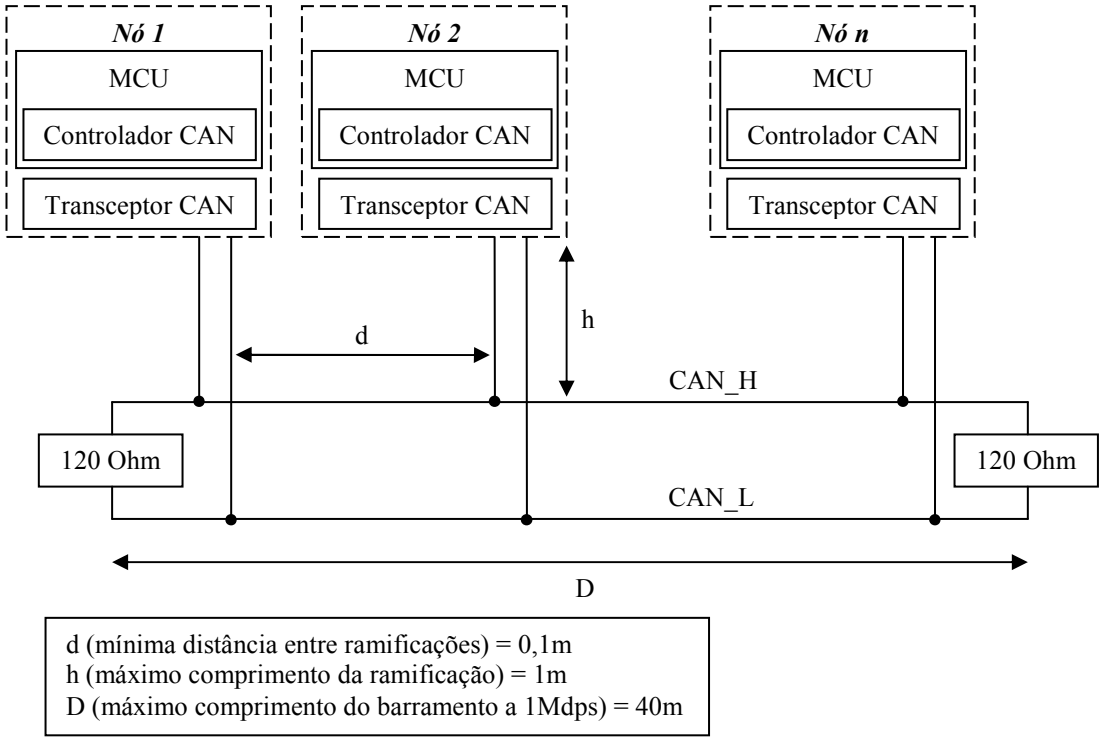


**Figura 2.1.3.2: Diagrama dos elementos constituintes da rede CAN.**

### 2.1.4. Controladores CAN

As interfaces de controle CAN originais – *BasicCAN* – ofereciam um número limitado de *buffers* de recepção e filtros, tipicamente 1 a 3. Se um nó usando tal controlador necessitasse ler um número de mensagens diferentes (diferentes identificadores), os filtros normalmente tinham que ser ajustados para *wide open* causando uma interrupção com toda

mensagem presente no barramento. Obviamente, o MCU iria realizar muitas interrupções CAN, então ele teria que checar em software se a mensagem poderia ser ignorada ou precisaria ser trabalhada.

As interfaces *FullCAN* tem uma quantidade de mensagens objeto, tipicamente 15, cada uma bi-direcional (configurável como transmissor ou receptor), tendo seu próprio *buffer* e filtro. Isto permite ajustar uma mensagem objeto para reconhecer somente um identificador. Enquanto o número total de mensagens em um nó que precisam ser lidas for menor que o número de mensagens objeto disponíveis, estas interfaces são muito eficientes. Elas só causam uma interrupção no MCU se uma mensagem desejada for recebida.

Entretanto, este mecanismo oferece nenhuma proteção contra um cenário pior de mensagens consecutivas – *back-to-back*. Cada mensagem objeto tem um único *buffer* e uma mensagem recebida sobrescreverá o conteúdo do *buffer*, então potencialmente mensagens podem ser perdidas. Enquanto o *buffer* é configurado para um único identificador, este cenário não é tão ruim, pois é improvável que o transmissor produza-as em sequência. Porém, o MCU precisa estar configurado para receber vários identificadores e isso implica, em 1Mbps, cerca de 50us entre receber a ocorrência de interrupção e uma mensagem ser sobrescrita pela próxima.

O único meio de contornar problema de *back-to-back*, o alto desempenho e a demanda de tempo numa rotina de interrupção é com *buffer* FIFO – *First In - First Out*. As características típicas destas contemplam um número de filtros que incluem registradores de máscara – *mask register* e de correspondência – *match register*. Após o *match register*, a mensagem recebida é movida para o FIFO. Uma interrupção é feita ao MCU dependendo da configuração: certo nível de ocupação é alcançado ou a alta prioridade do filtro da última mensagem recebida.

Mesmo que um FIFO mantenha 64bytes, é ainda grande o suficiente para melhorar o *back-to-back*. Se o FIFO é configurado para causar uma interrupção com toda mensagem recebida, o MCU tem no mínimo 500 *bit time* até o FIFO entrar em *overflow*. Isto é cerca de 10 vezes mais tempo disponível para o MCU do que com *BasicCAN* ou *FullCAN*. No entanto, mensagens no FIFO não podem ultrapassar uma as outras. Deste modo, se o FIFO tem várias mensagens e uma de maior prioridade entra, o MCU precisa processar as anteriores antes de acessá-la. Numa interface *FullCAN* é possível que a rotina de interrupção permitisse que a mensagem de alta prioridade ultrapassasse as de menor prioridade.

Os fabricantes de *chips* já oferecem dispositivos que combinam os avanços do *FullCAN* com FIFO. A abordagem mais poderosa é uma FIFO dedicada para cada uma das mensagens objetos. Porém, estes são os controladores mais complexos para configurar, especialmente se cada FIFO individualmente pode ser livremente localizada na RAM. Outra alternativa é concatenar as mensagens objeto para o FIFO, então ao invés destas terem somente um *buffer*, o FIFO pode ser formada por "empréstimos" de *buffers* de outras mensagens objeto. Embora flexível, há a desvantagem de cada *buffer* adicionado ao FIFO, uma mensagem objeto é perdida. Assim, o valor desta *feature* aumenta com o maior número de mensagens objeto, mas decresce se o número destas diminui. Contudo, a escolha correta da interface CAN depende de uma análise de pior caso para a aplicação que ela operará [13].

### 2.1.5. Tratamento de erros no CAN

As técnicas sofisticadas de retransmissão e de detecção de erros numa rede CAN provem um alto nível de integridade de dados, que em muitas aplicações é crucial. Estes mecanismos são de nível de *bits*, de mensagens e físico. No nível de *bit* pode haver duas situações de erro:

- *Bit monitoring*: O módulo transmissor verifica o estado do barramento depois da escrita de um *bit* dominante. Se lido um *bit* recessivo é sinal de erro;
- *Bit Stuffing*: permite-se a escrita de apenas cinco *bits* consecutivos de mesmo valor. Acrescenta-se um *bit* de valor oposto, *bit stuff*, entre os grupos de *bits* iguais. O receptor é encarregado de retirá-lo, e no caso de receber pelo menos seis *bits* semelhantes consecutivos é sinal de erro. Isto permite interromper níveis DC sobre o barramento e disponibilizar bordas nos *bits* de dados, os quais são usados para re-sincronização.

No nível de mensagem são três possíveis erros:

- CRC: O módulo transmissor calcula um valor conforme os *bits* da mensagem e envia-o com ela. O módulo receptor recalcula e compara com o CRC recebido. Este campo pode detectar e corrigir quatro *bits* de erro na região do SOF ao início do CRC;
- *Frame Check*: Alguns *bits* da mensagem recebida são constantes e determinados pelo padrão CAN;
- *Acknowledgment Error Check*: O módulo transmissor coloca um *bit* recessivo no campo ACK, e o(s) módulo(s) responde(m) ao transmissor colocando um *bit* dominante no campo ACK, a cada mensagem íntegra recebida, independentemente do resultado do teste de aceitação. Caso o módulo transmissor original da mensagem não receba resposta indica que a mensagem estava corrompida ou nenhum módulo a recebeu.

Contudo, a ocorrência de toda e qualquer uma destas falhas faz com que um ou mais módulos receptores coloquem mensagem de erro no barramento – *Error Frame*, o qual é formado por seis *bits* dominantes consecutivos. Isto permite qualquer controlador avisar toda

a rede de um erro logo que ele é identificado sem ter que aguardar o fim da mensagem, além de sinalizar ao transmissor que ele deve reenviar a mensagem.

Há também um contador de erros que é incrementado em uma unidade nos módulos receptores e em oito nos módulos transmissores a cada mensagem transmitida ou recebida erroneamente. São considerados normais os módulos com contadores zerados, no modo *Error Active* os com contadores entre 1 e 127 e em *Error Passive* os com contadores entre 128 e 255. Módulos no modo *Error Passive* respondem a *Error Frames*, porém se gerarem um *Error Frame* escrevem *bits* recessivos no lugar de dominantes. Os módulos com contadores maiores que 255 são considerados em *Bus Off* e não atuam no barramento, necessitando do MCU para reinicializarem o controlador CAN na rede. Os contadores são decrementados conforme as mensagens íntegras são recebidas.

No nível físico, caso ocorra curto do CAN_H ou CAN_L a GND ou VCC, curto entre CAN_H e CANL ou ruptura de um destes a rede continua operando numa espécie de modo de segurança.

### 2.2. Especificações do Projeto

A rede de comunicação CAN proposta é simples, compõe-se de duas ECUs conectadas. Essa topologia é útil para simular o correto envio/recebimento de mensagens pelas ECUs. Ainda que uma rede CAN, geralmente, tenha mais nós, a principal diferença encontrada nos dois casos, seria quanto à configuração dos identificadores, cujo valor é o que teoricamente determina a quantidade de nós, no caso de incremento/decremento destes. Porém, essa diferença é irrelevante do ponto de vista do funcionamento da rede em si, isto é, no reconhecimento dos nós e a na comunicação entre eles, além de que uma das características da CAN é sua possibilidade de expansão. Portanto, o principal aspecto a ser analisado é a correta troca de mensagens entre os nós.

Neste projeto é proposto que cada uma das ECUs leia os dados de entrada provenientes de seus sensores, empacote-os no devido formato e os envie ao outro nó de rede. Neste nó, a correspondente ECU deve processar a mensagem e atuar, se necessário, da forma solicitada.

É necessária uma interface de diagnóstico ligada a uma ou as duas ECUs, com o objetivo de acompanhar o funcionamento correto da rede. Para a aplicação proposta no presente projeto, representada Figura 2.2.1, a interface de diagnóstico é ligada a somente uma ECU e realiza o acesso ao conteúdo de somente esta.

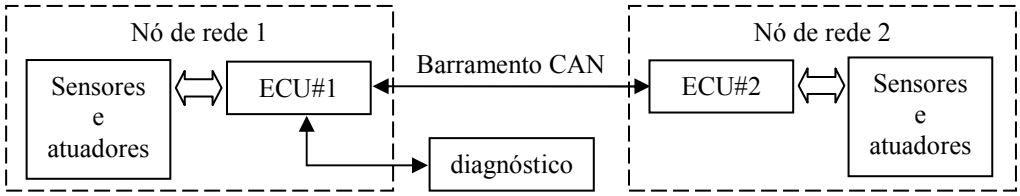


**Figura 2.2.1: Representação em blocos da aplicação proposta.**

# 3. ANÁLISE DE ALTERNATIVAS

Os elementos escolhidos para realizar a rede CAN proposta no Capítulo 2 (ver Figura 2.2.1) são um sistema microcontrolado com controlador CAN integrado – que desempenha o papel de nó de rede 1, um sensor com interface de rede CAN – que desempenha o papel de nó de rede 2 e um terminal PC – *Personal Computer* conectado somente a uma ECU com comunicação serial RS232, cuja função é de visualização do conteúdo das mensagens trocadas entre os dois pontos da rede.

O protótipo nó do presente projeto foi baseado na placa de desenvolvimento McBoard com bandeira LPC2368 desenvolvida pela Labtools. Enquanto que o dispositivo sensor é o A*bsolute Encoder* BMMH 42S1N24B12/18P25 fabricado pela Baumer Electric AG. A aplicativo da interface de visualização e acesso das mensagens é o HiperTerminal, Private Edition, Versão 6.3.

A Figura 3.1 representa a aplicação proposta com os elementos escolhidos. A estrutura apresentada foi definida em função dos dispositivos e tem por objetivo validar o desenvolvimento do nó conectado a um dispositivo de rede CAN.

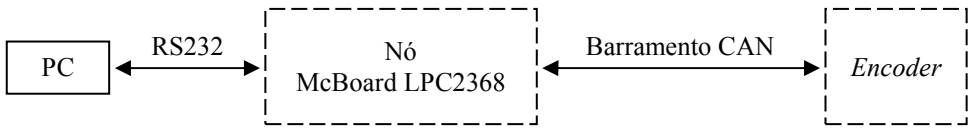


**Figura 3.1: Representação em blocos dos elementos do projeto.**

## 3.1. McBoard LPC2368

A placa McBoard LPC2368, Figura 3.1.1, oferece os recursos necessários para a realização do nó com comunicação CAN. Entre outros [15], cita-se:

- Microcontrolador LPC2368;
- Comunicação CAN;

- Comunicação serial RS232;
- Entrada analógica – AIN;
- Saída analógica – AOUT;
- Teclas e leds (4 convencionais e 1 RGB).

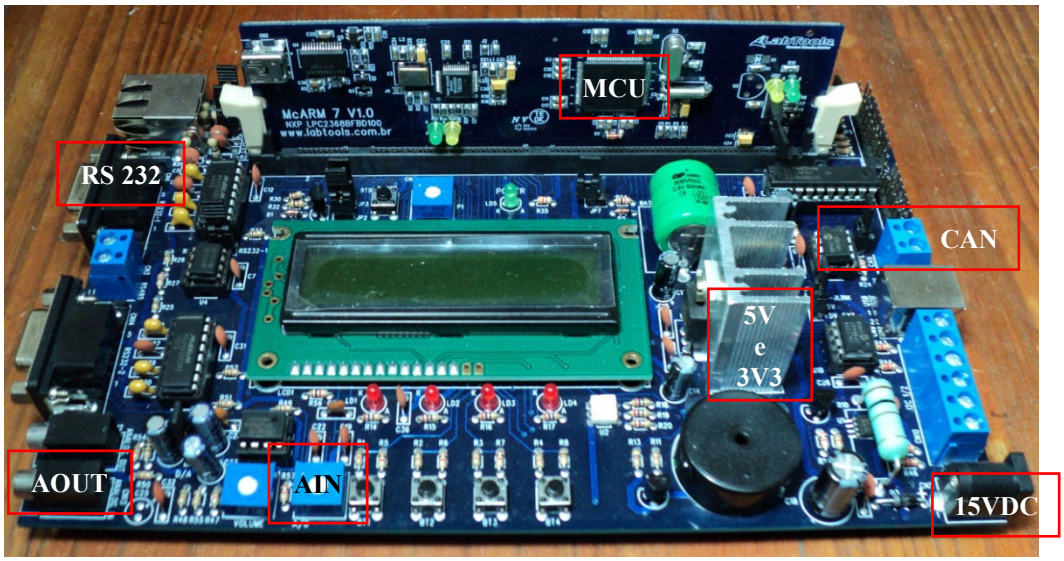


**Figura 3.1.1: Foto da placa McBoard LPC2368, em destaque o MCU e alguns dos periféricos utilizados no projeto.**

A placa é alimentada por 15VDC e dispõe de reguladores de 5V e 3,3V para os seus periféricos.

### 3.1.1. Microcontrolador LPC2368

O MCU escolhido para o desenvolvimento é LPC2368 da NXP Semiconductors. Este MCU é baseado numa CPU de 16-*bit*/32-*bit* ARM7TDMI-S.

A arquitetura ARM – *Advanced RISC Machine* proporciona uma máquina RISC e tem a filosofia de ser simples. O ARM7 é um computador RISC com uma pequena série de instruções, tornando-o ideal para sistemas embarcados. O coração do ARM7 CPU é

*instruction pipeline* de três estágios independentes de *hardware* – executa uma instrução enquanto decodifica a segunda e recupera a terceira da memória.

O ARM7 é uma arquitetura *load-and-store*, as instruções são executadas por 16 registradores centrais de 32 *bits* (R0 à R12 são *user registers*, R13 é o *stack pointer*, R14 é o *link register* e R15 é o *program counter*). Há ainda o CPSR – *Current Program Status Register* de *flags* as quais relatam e controlam as operações do ARM7 CPU.

O ARM7 CPU é capaz de executar duas séries de instruções, *ARM instruction* de 32 *bits* e *THUMB instruction* de 16 *bits*. Este é uma forma comprimida do *ARM instruction*, embora resulte em um desempenho menor comparado ao *ARM instruction*, ele permite uma densidade maior de código.

O ARM7 CPU possui sete diferentes modos de operação: *system & user*, FIQ, *supervisor*, *abort*, IRQ e *undefined*, cada um associado a um vetor de interrupção – endereço. A aplicação opera em *system & user*, porém caso ocorra uma exceção (tal como uma interrupção, *reset* ou erro de memória) o processador muda o modo de operação. Cada modo de exceção possui um registrador adicional denominado SPSR – *saved program status register*, o qual armazena o valor de CPSR antes de sair de *system & user*. Quando um modo de exceção é executado, exceto FIQ, os registradores R0 a R12 mantém-se os mesmos, porém o R13 e R14 são substituídos por outros próprios para cada modo. O retorno para *system & user* segue uma sequência própria a cada modo.

O núcleo do ARM7 conecta-se aos periféricos por três barramentos principais, um barramento local para o acesso a portas de alta velocidade, a memória RAM e FLASH, o AHB – *Advanced High Performance Bus* (de alta velocidade, reservado para VIC, USB e Ethernet) e o APB – *Advanced Peripheral Bus* (para os demais periféricos). Os pinos do microcontrolador são multiplexados e sua função padrão é de propósito geral.

A versão ARM7TDMI é amplamente utilizada na indústria e este MCU está presente em aplicações como controle industrial, sistemas médicos, conversores de protocolos e comunicações [14], [17].

As suas principais características [16], pertinentes ao projeto, do LPC2368 são:

- Alimentação de 3,0 a 3,6VDC;
- 512kbytes de memória flash de alta velocidade;
- 32kbytes de memória volátil RAM;
- 32 interrupções (6 externas);
- Controlador CAN com dois canais;
- 4 UARTs;
- 1 conversor A/D de 10*bits* com 4 canais disponíveis;
- 1 conversor D/A de 10*bits*;
- Opera com cristal de 1MHz a 25MHz;
- 72MHz de operação máxima via PLL interno.

### 3.1.2. Comunicação CAN

A placa possui o *transceiver* MCP2551 conectado ao CAN2 do microcontrolador. Através do conector CN1 a comunicação é estabelecida por meio de duas vias, CAN_H e CAN_L.

O bloco CAN do MCU é projetado para suportar múltiplos barramentos simultaneamente, permitindo ao dispositivo atuar como *gateway*, *switch* ou *router*. O módulo CAN consiste do controlador e do filtro de aceitação. Todos registradores e a RAM são acessadas como palavras de 32-*bit*. As características do controlador CAN são [16]:

- Duplo *buffer* de recepção e triplo *buffer* de transmissão;
- Taxa de dados de até 1Mbps em cada canal;

- Registradores de 32 *bits* e acesso a RAM;
- Compatibilidade com CAN2.0B;
- Filtros de aceitação de identificadores reconhecem identificadores de 11-*bit* e 29-*bit* para todos os canais;
- Filtro de aceitação permite recepção automática *FullCAN* para identificadores padrões selecionados.

Os registradores envolvidos no recebimento de mensagens pelo controlador são: *Receive Identifier Register* – RID responsável pelo ID, *Receive Data Register* A – RDA responsável pelos *bytes* B1 ao B4 e *Receive Data Register* B – RDB responsável pelos *bytes* B5 ao B8.

Os registradores envolvidos na transmissão de mensagens pelo controlador são: *Transmit Identifier Register* – TID responsável pelo ID, *Transmit Data Register* A – TDA responsável pelos *bytes* B1 ao B5 e *Transmit Data Register* B – TDB responsável pelos *bytes* B5 ao B8.

### 3.1.3. Comunicação serial RS232

A placa possui dois *transceivers* MAX3232 para adequar os níveis de tensão do MCU ao padrão RS232C (+12V e -12V). Utiliza-se o conector RS232-1, conectado a UART0, para a comunicação, a qual é feita por meio de duas vias, TX1 e RX1.

O bloco de *Universal Asynchronous Receiver/Transmitters* – UART do MCU apresenta FIFOs de transmissão e recepção de 16 bytes [16].

### 3.1.4. Entrada analógica – AIN

A entrada analógica do MCU (AD0 – A/D *converter* 0, *input* 0) é interligada a um *trimpot* na placa. O AD0 é um conversor de 10 *bits* por aproximação sucessiva com unidade

S&H, com range de medidas de 0 a 3V (resolução de amplitude de, aproximadamente, 3mV) e tempo de conversão de no mínimo 2,44us (o que limita a aquisição de sinais até, aproximadamente, 200kHz).

### 3.1.5. Saída analógica – AOUT

A saída analógica do MCU (AOUT – D/A *converter output*) é interligada na placa ao conector CN12 (ANALOG OUT). O AOUT é um conversor de 10 *bits* e seu tempo de conversão depende de seu registrador interno, da impedância e capacitância externas, podendo ser no mínimo 1us, em determinada condição.

### 3.1.6. Teclas e LEDs

A placa possui quatro leds vermelhos, um led RGB (*Red*, *Green* e *Blue*) e quatro teclas de contato. Todos são ligados ao MCU por meio de GPIO – *General purpose digital input/output pin*, os quais têm sua configuração controlada por *bits* individuais.

## 3.2. *Encoder*

O *Absolute Encoder* BMMH 42S1N24B12/18P25, Figura 3.2.1, é um transdutor industrial utilizado para medir ângulos e rotações. Algumas de suas características são [18]:

- Alimentação de 10 a 30VDC;
- Tempo de inicialização de 170ms após ligado;
- Resolução multivoltas de 18-bit;
- Interface integrada com CANopen;
- Alta resistência a impactos e vibrações;
- Resolução e ponto zero programável.

**Figura 3.2.1: Foto do *Encoder* com controlador CAN integrado utilizado no projeto.**

O *Encoder* possui cinco terminações com cabeamento de cores diferentes para alimentação e comunicação.

### 3.2.1. CANopen

O CANopen é um protocolo de alto nível otimizado para troca rápida de dados em sistemas de tempo real. Este permite acesso simplificado a todos dispositivos e parâmetros de comunicação da rede, sincronização, configuração automática da rede e processo de comunicação de dados cíclico ou por evento controlado.

A comunicação baseia-se em quatro *communication objects* (COB) com características diferentes:

- *Process data objects* – PDO para dados de tempo real;
- *Service data objects* – SDO para transmissão de configuração e parâmetro;
- *Network management* – NMT para inicialização e controle da rede;
- Objetos para sincronização – SYNC e mensagens de emergência – EMCY.

Todos os parâmetros de comunicação e dispositivos são subdivididos em um diretório objeto. Um diretório objeto inclui o nome do objeto, tipo de dado, número de subíndices, estrutura dos parâmetros e o endereço [19].

A estrutura das mensagens CANopen, mostrada na Figura 3.2.1.1, é composta por um campo identificador de 11 *bits* denominado COB ID, seguido pelo DLC e por oito *bytes*.

| COB ID | DLC | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte 7 | Byte 8 |
|---|---|---|---|---|---|---|---|---|---|
| Xxx | x | xx | xx | xx | xx | xx | xx | xx | xx |

**Figura 3.2.1.1: Estrutura da mensagem CANopen.**

O COB ID possui duas partes, a primeira de quatro *bits* é o *Function code* que prove informação do tipo de mensagem e prioridade, enquanto os sete *bits* seguintes são relacionados ao nó.

| Function code | Node ID |
|---|---|
| 4-bit function code | 7-bit node ID |

**Figura 3.2.1.2: Estrutura do COB ID.**

# 4. MÉTODOS, PROCESSOS E DISPOSITIVOS

A análise de redes CAN e a definição dos elementos a serem utilizados no projeto foram necessárias para estabelecer o seu escopo. A partir disso, iniciaram-se novas etapas onde a construção *firmware* de controle do nó foi o ponto de maior empreendimento. A Figura 4.1 representa um diagrama de blocos do projeto.

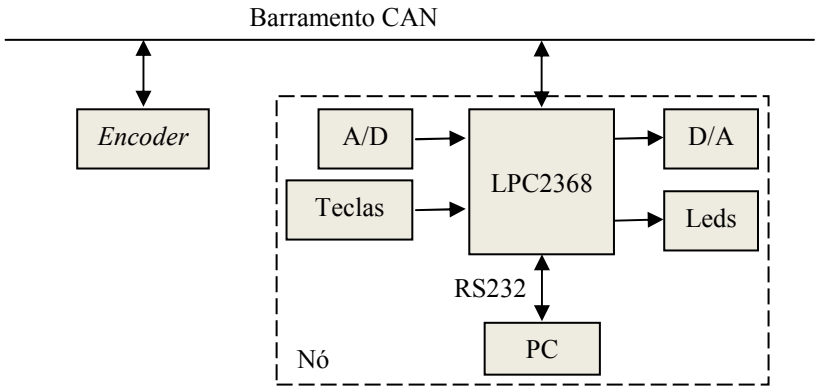


**Figura 4.1: Representação em blocos da rede entre o *encoder* e o nó, com seus elementos internos.**

O desenvolvimento do projeto seguiu basicamente duas etapas:

- *Firmware* para o MCU LPC2368;
- Definição do dicionário de dados;
- Testes em hardware com o *encoder*.

## 4.1. *Firmware*

O ambiente de desenvolvimento utilizado para a criação do *firmware* foi uma versão *evaluation* do µVision V4.03 da Keil Elektronik GmbH/Keil Software, Inc. 1995 – 2009.

### 4.1.1. Introdução

O desenvolvimento do *firmware* para o ARM7 iniciou-se com códigos de testes dos periféricos usados organizados em versões. Cada uma delas tinha o objetivo de validar por meio de simulações no µVision ou em *hardware* a função de configuração ou execução a ser utilizada na versão final. Ao passo que as versões eram validadas, as suas funções eram adicionadas às versões posteriores quando necessário. Citam-se algumas destas versões e o(s) aspecto(s) testado(s):

- Versão III: *Heartbeat* – led piscando e *clock* interno/externo;
- Versão IV: AD/DA – leitura e escrita;
- Versão V: UART – envio e recepção de mensagens;
- Versão VII: CAN – comunicação em *self-test*;
- Versão X: IRQ – modo de operação e interrupção VIC;
- Versão XI: eCODE – código estruturado;
- Versão XIII: hLED – LED em *hardware*;
- Versão XIV: hUART – UART0 em *hardware*;
- Versão XV: hCAN – CAN em *hardware*.

#### 4.1.1.1. *Heartbeat*

Este código faz um led piscar a uma freqüência aproximada de 1Hz, após um *delay* de 10s do *reset*. Foi realizada a inclusão do *header file* referente à família LPC23xx, a edição do *startup*, acesso aos registradores do MCU para inicialização do PLL interno, escolha como fonte principal de *clock* o oscilador interno ou o externo, configuração de pino como saída ou entrada e a alteração de seu estado (“0” e “1”).

#### 4.1.1.2. AD/DA

Este código realiza a leitura do AD e a escrita no DA. Foram construídas funções de acesso aos registradores de seleção de função dos pinos, aos registradores de modo de operação dos conversores e aos registradores de acesso à leitura e escrita.

#### 4.1.1.3. UART

Este código realiza a comunicação entre um terminal serial e as duas UARTs do MCU. Foram construídas funções de acesso aos registradores de *power control* de periféricos, aos registradores de *buffer* de transmissão e recepção da UART e de configuração do *baud rate*.

#### 4.1.1.4. CAN

Este código realiza a troca de mensagens dos dois canais CAN do MCU. Foram construídas funções de acesso aos registradores pertinentes ao funcionamento da comunicação CAN, são eles: registradores de *buffer* de transmissão e recepção, de modos de operação, de filtros de aceitação e de configuração do *baud rate*.

#### 4.1.1.5. IRQ

Este código estabelece o funcionamento do MCU quando na presença de interrupções externas. Foram construídas funções configuração de *Vectored Interrupt Controller* – VIC para *Interrupt Request* – IRQ.

#### 4.1.1.6. eCODE, hLED, hUART e hCA

eCODE é a edição da estrutura final do *firmware*. Os outros códigos são preparados para realizar os testes na McBoard.

### 4.1.2. Versão Final

A versão final do *firmware* do projeto é representada conforme os fluxogramas da Figura 4.1.2.1 a Figura 4.1.2.4.

A Figura 4.1.2.1 representa a inicialização microcontrolador após um *reset* ou um *power on*. A função "message_initial" envia pela UART0 a lista de opções disponíveis, todos os itens se referem à comunicação CAN. "COB CAN" é usada para enviar uma mensagem, "lê bus" executa a leitura dos *buffers* de recepção do controlador CAN, "release" torna o conteúdo dos *buffers* de recepção passíveis a substituição, "clear Rx" apaga o conteúdo dos *buffers* de recepção, "abort Tx" cancela o pedido de transmissão do *buffer* de transmissão e "position" é uma mensagem CANopen que requisita ao *encoder* a sua posição.

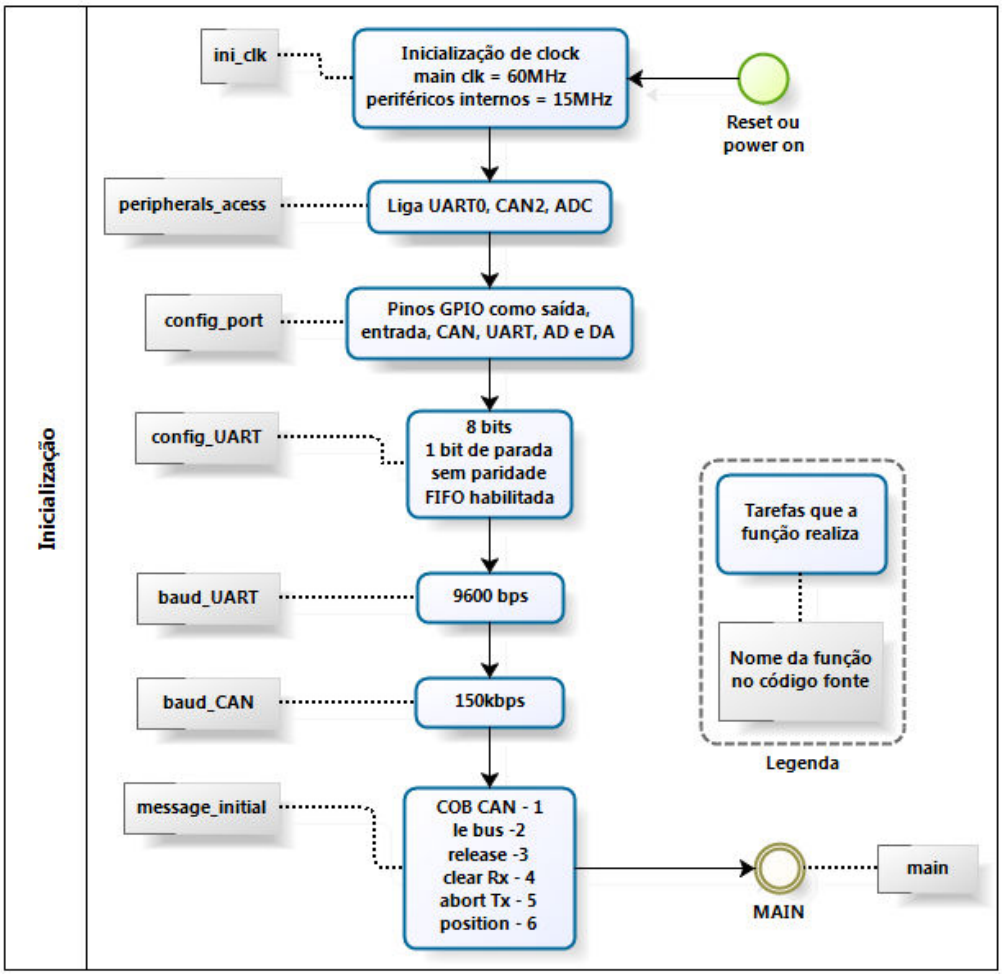


**Figura 4.1.2.1: Fluxograma da inicialização do *firmware*.**

A Figura 4.1.2.2 representa a função “main” do código. Há uma rotina de leitura do AD0, escrita do valor no AOUT e verificação de recebimento de caractere pela UART. No caso de receber algum pertencente àqueles mostrados na mensagem inicial, é acessada a função “U0_function”.

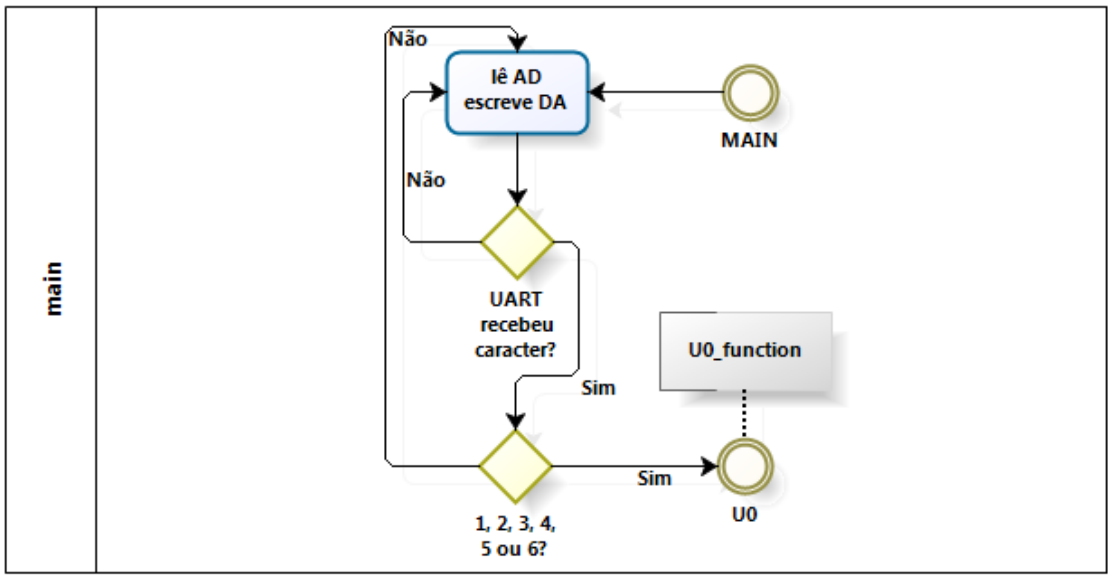


**Figura 4.1.2.2: Fluxograma da função main do *firmware*.**

A Figura 4.1.2.3 representa a função “U0_function” do código. Esta rotina verifica a opção escolhida e configura as suas ações. No caso de “COB CAN” é pedido o “ID_length” que é o tipo de ID (11 ou 29 *bits*), o ID, RTR, length (número de *bytes* a ser transmitidos) e *data* que são os *bytes* de dados. Esses valores são gravados em uma *struct*.

A Figura 4.1.2.4 representa a função “CAN_send” do código. Os valores anteriormente gravados na *struct* são alocados nos registradores de transmissão do CAN e dado o pedido de envio da mensagem.

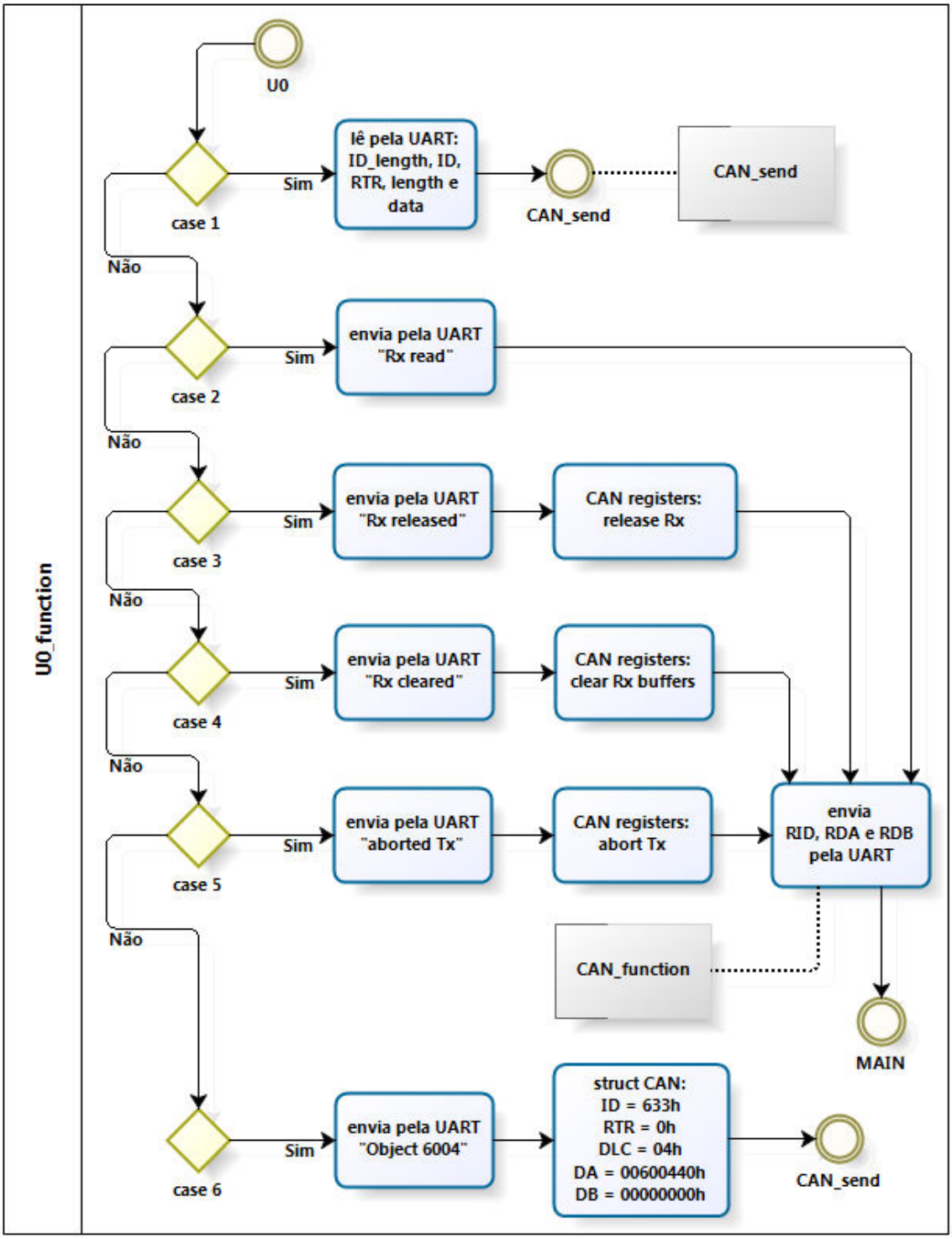


**Figura 4.1.2.3: Fluxograma da função U0_function do *firmware*.**

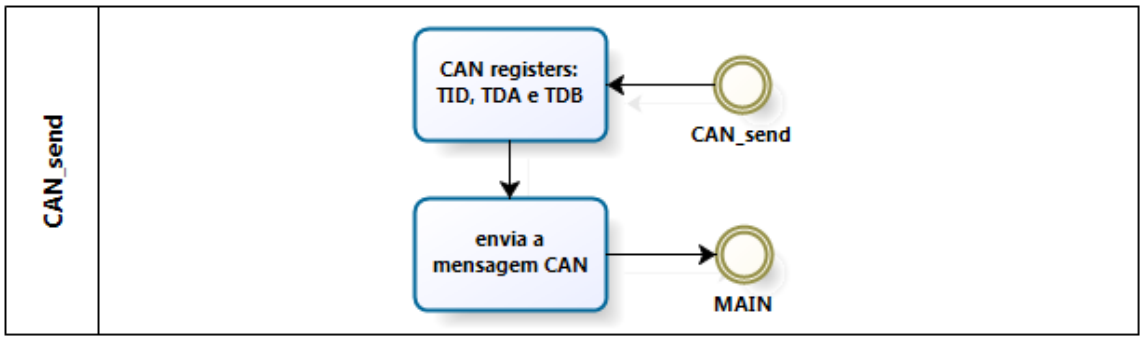


**Figura 4.1.2.4: Fluxograma da função CAN_send do *firmware*.**

## 4.2. Dicionário de dados

O *encoder* utilizado baseia-se em comunicação CANopen. O padrão de mensagens utilizadas para a comunicação entre este e o nó é definido como Dicionário de dados. O Dicionário de dados (DD) da aplicação proposta, conforme a Tabela 4.1.2.1, parte das informações referentes às mensagens COB e limita-se a algumas das possíveis mensagens suportadas pelo *encoder* [19], o qual tem nó ID igual a 33h.

| Mensagens ponto a ponto | | |
|---|---|---|
| *Function Code* | COB ID | COB ID |
| Emergência | 80h + nó ID | B3h |
| PDO1 (tx) | 180h + nó ID | 1B3h |
| PDO2 (tx) | 280h + nó ID | 2B3h |
| SDO (tx) | 580h + nó ID | 5B3h |
| SDO (rx) | 600h + nó ID | 633h |
| Mensagens *Broadcast* | | |
| *Function Code* | COB ID | |
| NMT | 0 | |
| SYNC | 80h | |

**Tabela 4.2.1: COB ID de mensagens (rx e tx do ponto de vista do *encoder*) estabelecidas no dicionário de dados do projeto.**

### 4.2.1. SDO

As configurações do *encoder* são baseadas em diretórios objetos e objetos. É possível acessá-los por meio de índice e subíndice. Os dados podem ser requisitados ou escritos dentro de um objeto, quando aplicável.

| COB ID | DLC | Command | Object L | Object H | Subindix | Data 0 | Data 1 | Data 2 | Data 3 |
|---|---|---|---|---|---|---|---|---|---|

**Figura 4.2.1.1: Estrutura do SDO.**

O COB ID de um SDO é composto por 633h (600h + nó ID) – mensagem do nó para o *encoder* e 5B3h (580 + nó ID) – mensagem do *encoder* para o nó. O Command define se os dados são lidos ou ajustados e como os *bytes* de dados estão envolvidos.

| *Command* | Descrição | Data Length | |
|---|---|---|---|
| 22h | Requisição de *download* | Max. 4 *bytes* | Transmite parâmetro ao *encoder* |
| 23h | Requisição de *download* | 4 *bytes* | |
| 40h | Requisição de *upload* | - | Requisita parâmetro ao *encoder* |
| 42h | Resposta de *upload* | Max. 4 *bytes* | Parâmetro ao mestre com até 4 *bytes* |
| 43h | Resposta de *upload* | 4 *bytes* | |
| 60h | Resposta de *download* | - | Confirma recebimento ao nó |

**Tabela 4.2.1.1: Relação de alguns SDO *command* utilizados no DD.**

Um SDO usado no projeto é uma requisição de posição.

### 4.2.2. PDO

As mensagens PDO são usadas para troca de mensagens em processos em tempo real, como por exemplo, posição e status de operação. PDOs podem ser transmitidos assincronamente (ciclicamente) ou sincronamente.

### 4.2.3. NMT

As mensagens NMT, Figura 4.2.3.1, são usadas em serviços para monitoramento do dispositivo, usuários do barramento podem ser inicializados, postos em operação e parados. Além disso, o NMT é utilizado para monitoramento da conexão.

| COB ID = 0 | Command | Número do nó |
|---|---|---|

**Figura 4.2.3.1: Estrutura de mensagem NMT.**

Após a inicialização, o *encoder* envia ao barramento uma mensagem de *BootUp*, que contém o seu nó ID (COB ID = 700h + nó ID) e em seguida entra em modo pré-operacional. Neste estado, parâmetros SDO podem ser lidos e escritos. Para requisitar parâmetros PDO, o *encoder* deve primeiro ser movido ao estado de modo operacional por meio de um *command byte* 01h, conforme mostrado na Figura 4.2.3.2. Outros *command byte* podem ser usados para alterar o estado do *encoder*, conforme a Tabela 4.2.3.1.

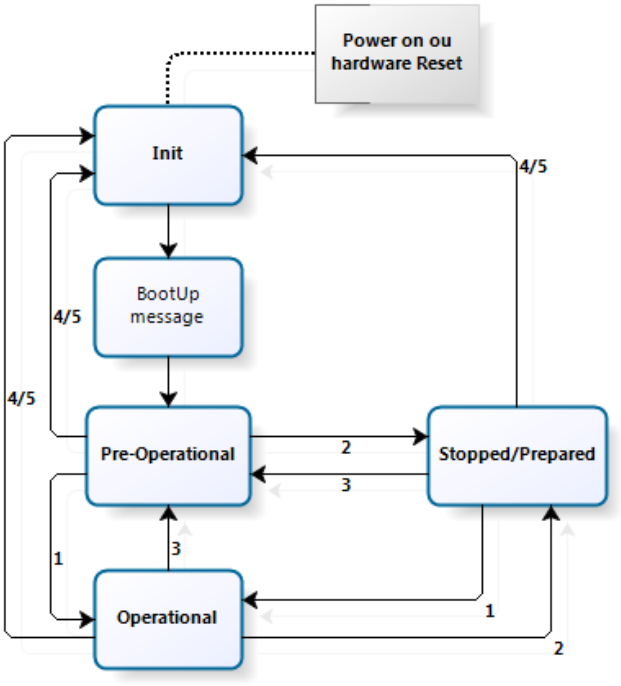


**Figura 4.2.3.2: Representação de possíveis estados do *encoder*.**

| *Command byte* | Descrição | Estado na Figura 4.2.3.2 |
|---|---|---|
| 01h | Inicia nó remoto | 1 |
| 02h | Pára nó remoto | 2 |
| 80h | Entra em modo pré-operacional | 3 |
| 81h, 82h | *Reset* nó remoto | 4, 5 |

**Tabela 4.2.3.1: Valores de *Command byte* e o correspondente estado do *encoder*.**

# 5. RESULTADOS ALCANÇADOS

Os experimentos foram realizados no Laboratório de Instrumentação Eletro-Eletrônica (IEE) do Departamento de Engenharia Elétrica da Escola de Engenharia da UFRGS.

O primeiro aspecto a ser destacado entre os resultados encontrados é o desenvolvimento do *firmware* de controle do microcontrolador o que possibilitou a comunicação do nó a um elemento de barramento CAN, o segundo é a correta comunicação da rede, isto é, as mensagens trocadas e o efeito destas no *encoder* e o último é o desenvolvimento da interface genérica.

## 5.1. Rede CAN

O *firmware* desenvolvido para a plataforma ARM7 torna possível o funcionamento do nó. A possibilidade de manipulação de variáveis analógicas e digitais de entrada e a atuação em suas saídas é o esperado de uma ECU, a qual tem sensores e atuadores.

O nó ao comunicar-se com o dispositivo CAN utilizado para simular a rede, *encoder*, tornou possível a validação do sistema, pois foi realizada a correta troca de mensagens entre os pontos da rede, Figura 5.1.1. A verificação disto foi permitida com o auxilio da interface de visualização e acesso de mensagens (comunicação serial RS232) e o conhecimento dos padrões de mensagens do *encoder*, no caso CANopen.

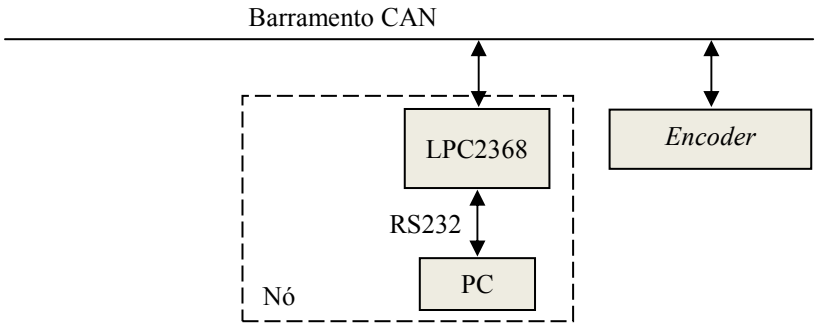


**Figura 5.1.1: Representação em blocos da rede CAN validada, nota-se o nó e o *encoder*.**

## 5.2. Comunicação nó e *encoder*

A sequência de passos descreve o acesso ao *encoder* e a leitura de posições. Todos os comandos enviados para o MCU a partir do HyperTerminal são limitados às opções: COB CAN – 1, lê bus – 2, release – 3, clear Rx – 4, abort Tx – 5 e position – 6 (ver seção 4.1.2).

O primeiro passo para estabelecer comunicação numa rede CAN é conhecer o seu *baud rate*. O *encoder* estava configurado para 250kbps e isso foi verificado através da medida no osciloscópio Tektronix TDS 210 do período de tempo de um *bit* de sua mensagem de *BootUp*, Figura 5.2.1. Posteriormente, os registradores do MCU foram configurados para que o controlador CAN do nó trabalhasse em 250kbps também.

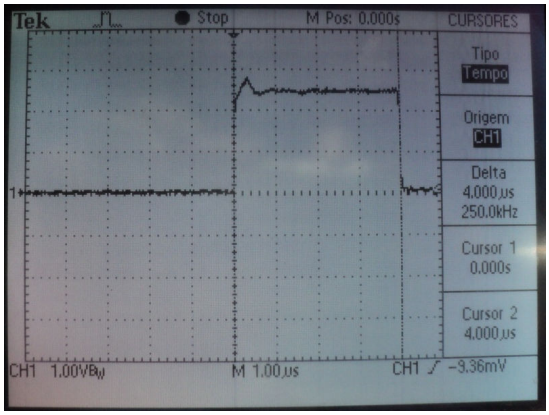


**Figura 5.2.1: Foto do osciloscópio medindo um *bit* da mensagem de *BootUp* do *encoder*.**

Após o *encoder* ser ligado, consequentemente enviar a mensagem de *BootUp*, é dado o comando de "lê bus – 2" e desta forma descobre-se o nó ID = 33h (733h – 700h) do *encoder*, Figura 5.2.2. Desta forma, os dois parâmetros essenciais para a comunicação estão disponíveis, o *baud rate* e o nó ID.

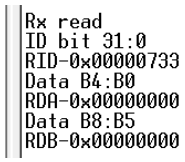


**Figura 5.2.2: Trecho do HyperTerminal de um comando de "lê bus" contendo a leitura dos registradores RID, RDA e RDB.**

A Tabela 5.2.1 mostra uma sequência de comandos enviados ao *encoder* quando este se encontra em estado pré-operacional. A coluna Nome da mensagem é também relacionada ao objeto acessado na EEPROM do *encoder* e as colunas de Parâmetros Digitados são os valores de entrada do *firmware* quando a opção "COB CAN – 1" é solicitada, onde a relação entre os *bytes* B8 a B1 e a estrutura do SDO são mostradas na Tabela 5.2.2. A última coluna referencia a mensagem com a correspondente visualização no HiperTeminal mostrada na Figura 5.2.2.

| Nome da mensagem | Parâmetros Digitados | | | | | Figura 5.2.2 |
|---|---|---|---|---|---|---|
| | ID 29/11 | ID | RTR | qtos bytes | B4:B1/B8:B5 | |
| *Baud rate* | 1 | 633 | n | 4 | 00**2100**40 | A |
| *Device name* | 1 | 633 | n | 4 | 00**1008**40 | B |
| Resolução (leitura) | 1 | 633 | n | 4 | 00**6001**40 | C |
| Resolução (escrita) | 1 | 633 | n | 8 | 00**6001**2300000800 | D |
| Resolução (leitura) | 1 | 633 | n | 4 | 00**6001**40 | E |
| Ler posição | 1 | 633 | n | 4 | 00**6004**40 | F |

**Tabela 5.2.1: Comandos enviados pelo nó ao *encoder*.**

| *byte* | Descrição | *Byte* | Descrição |
|---|---|---|---|
| B4 | *Subindex* | B8 | *Data 3* |
| B3 | Object H | B7 | *Data 2* |
| B2 | Object L | B6 | *Data 1* |
| B1 | *Command* | B5 | *Data 0* |

**Tabela 5.2.2: Relação entre os *bytes* B8 a B1 e os de uma mensagem SDO.**

Para todas as mensagens mostradas na Figura 5.2.2, após o seu envio é dado um comando de "lê bus – 2", uma vez que o *encoder* responde quando a recebe. Como se pode observar na Fig. 5.2.2 (a) é feita uma requisição de *upload* (*command* = 40h) de *baud rate* (*Object* = 2100h), posteriormente vê-se "5" em RDB que corresponde a 250kbps. Na Fig. 5.2.2 (b) é pedido o *Device name* e o valor hexadecimal em RDB convertido em ASCII e reorganizado é BMMx. Na Fig. 5.2.2 (c) é requisitado o valor da resolução e a resposta é de 64h, isto é, o *encoder* tem uma variação de 64h = 100 passos a cada 360º. Na Fig. 5.2.2 (d) é feita uma requisição de *download* (*command* = 23h) de *resolution* (*Object* = 6001), vê-se que RDA tem seu B0 = 60h, o que indica que o *encoder* confirma o recebimento, isto é, foi reconfigurado.

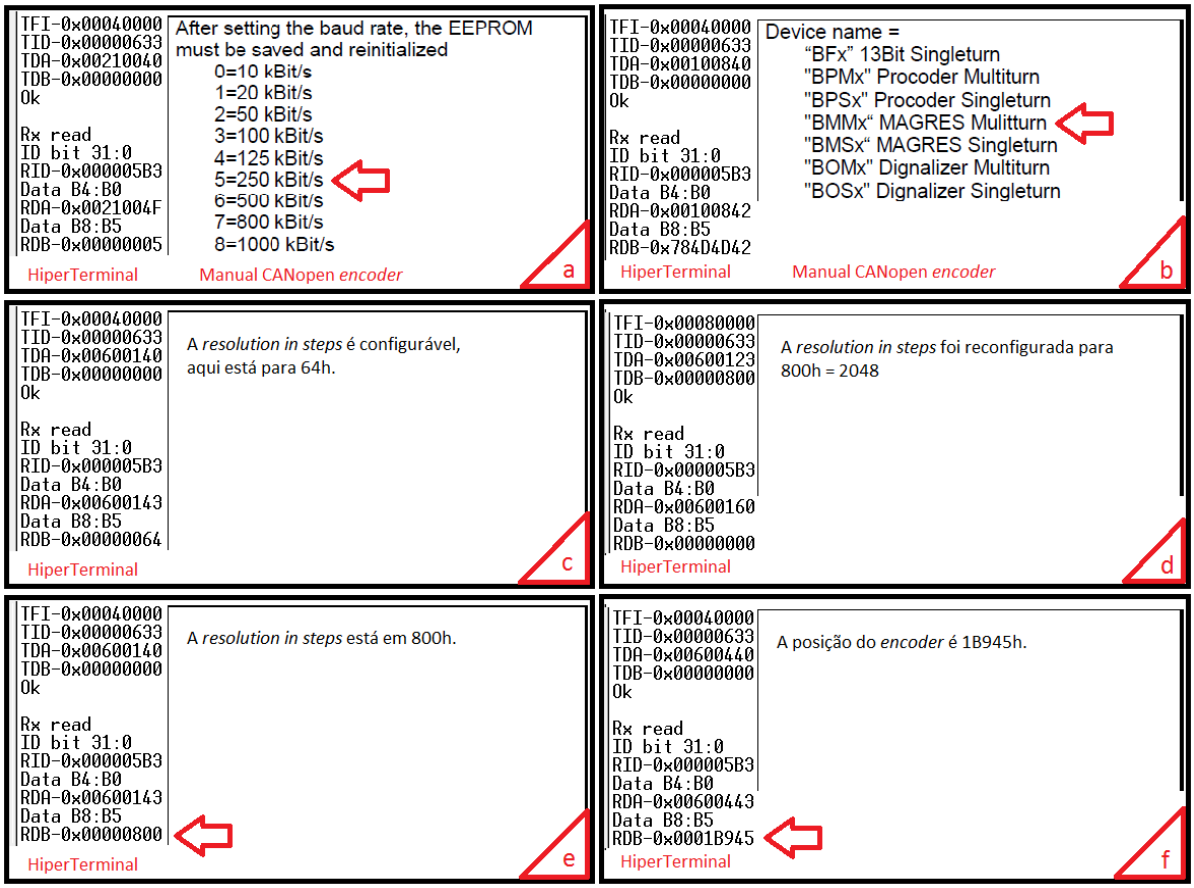


**Figura 5.2.3: Visualização do HiperTerminal conforme a mensagem da Tabela 5.2.1.**

Na Fig. 5.2.2 (e) é mostrado que efetivamente o *resolution* foi alterado com sucesso. Finalmente, na Fig. 5.2.2 (f) é realizada a leitura da posição atual do *encoder,* que é visto em RDB.

A Figura 5.2.3 mostra um ensaio realizado com o *encoder* ao ser variado, aproximadamente, em 180º.

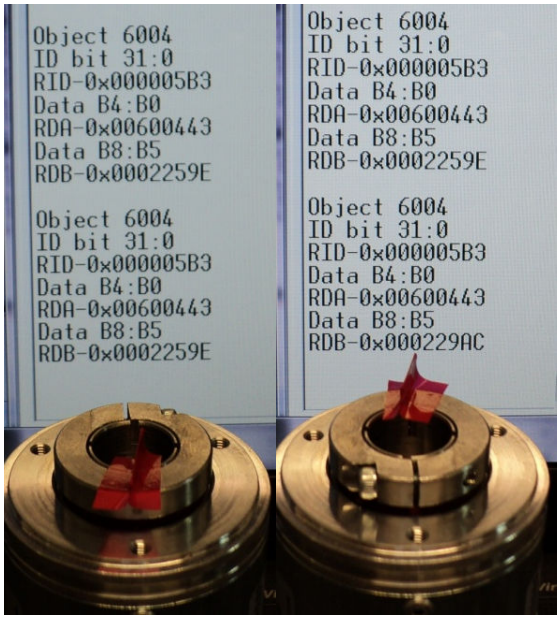


**Figura 5.2.4: Imagens do ensaio realizado com o *encoder*.**

A primeira leitura posição é de 2259Eh e a posterior é de 229ACh, desta forma a diferença é de 40Eh = 1038 e pela relação abaixo,

$$\frac{360°}{2048} \cdot (1038) \cong 182°$$

Tem-se aproximadamente 180º como esperado.

### 5.3. Interface genérica

Cabe destacar que a estrutura do *firmware* constitui-se de uma interface genérica, uma vez que este permite o nó estabelecer comunicação com dispositivos de outros protocolos de alto nível, não somente CANopen, desde que o dicionário de dados seja estabelecido. Há a

possibilidade também de ser construído um procotolo próprio, onde o DD seria desenvolvido conforme a aplicação em questão.

O nó desenvolvido foi testado na presença de somente uma ECU e na forma mestre-escravo, onde o *encoder* não requisitava parâmetros ao microcontrolador. Porém, numa rede onde o nó tivesse que receber e enviar parâmetros como, por exemplo, leitura de AD, escrita de DA e configuração de periféricos, alguns dos passos, em nível de *software*, deste para integrar-se seriam:

- Definição de um nó ID para mascarar as mensagens enviadas;
- Definição de filtros de aceitação para receber mensagens *broadcast*, *multicast* e ponto a ponto;
- Mascarar mensagens de envio com os dados requisitados – envio de dados em mensagens CAN;
- Mascarar mensagens de recepção com os dados recebidos – extração de dados em mensagens CAN.

Do ponto de vista do *hardware*, nenhuma mudança precisaria ser feita para adequar o nó a uma rede CAN.

# 6. CONCLUSÃO

Neste trabalho, apresentou-se o desenvolvimento de um *firmware* para uma plataforma ARM7 de um nó de redes CAN, por meio de uma placa de desenvolvimento comercial e um dispositivo de interface CANopen fizeram-se os testes de comunicação.

Inicialmente fez-se um estudo do protocolo CAN, posteriormente definiu-se a rede a ser constituída com o objetivo de validar o sistema CAN do nó. Utilizou-se como outro ponto da rede um dispositivo sensor do tipo *encoder* com interface CANopen integrada. Junto a ECU desenvolvida estabeleceu-se por meio de comunicação serial um sistema de monitoramento das mensagens enviadas pelo nó ao *encoder* e as respectivas respostas deste. Desta forma foi possível validar a implementação do canal CAN na plataforma ARM7 utilizada. A partir desta, pode-se estabelecer um nó que desempenhe a função de sensor e/ou atuador desde que sejam definidos os periféricos necessários e a sua conseqüente configuração no *firmware*.

A importância dos barramentos CAN pode ser facilmente percebida pelo volume de equipamentos que os utilizam e consequentemente a capacitação de recursos humanos também se faz necessário.

Como perspectiva de melhoria em relação à interface de visualização, o desenvolvimento de uma interface gráfica pode facilitar a manipulação das mensagens do CAN. Em relação ao nó, este pode ser potencializado para uso como *switch*, *router* ou *gateway* desde que o seu segundo canal (CAN1) seja disponibilizado. Com isso, pode-se ainda utilizá-lo como elemento de diagnóstico da rede, desta forma sua função seria de monitoramento de todas as mensagens presentes no barramento. Embora os dois canais estejam no mesmo microcontrolador, este poderia ser configurado para executar as duas tarefas sem prejuízo à operação do nó.

# REFERÊNCIAS


[1] Catsoulis, J. **Designing Embedded Hardware**. O'Reilly Media, 2005. p. 1-3. ISBN 978-0-596-55662-4.

[2] Heath, S. **Embedded Systems Design**. Newnes, 2003. p. 1-2. Disponível em: <http://books.google.com/books?id=BjNZXwH7HlkC&pg=PA2#>. Acesso em: 15 set. 2010.

[3] **CAN history**. Disponível em: <http://www.can-cia.org/index.php?id=161>. Acesso em: 22 set. 2010.

[4] Chaari, L.; Masmoudi, N.; Kamoun, L. **Electronic Control in Electric Vehicle Based on CAN Network**. In: IEEE Systems, Man and Cybernetics, n. 4, vol. 7, out. 2002. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1175668>. Acesso em: 6 out. 2010.

[5] Leteinturier, P. **Multi-core Processors: Driving the Evolution of Automotive Electronics Architectures**. Infineon Technologies. Disponível em: <http://www.eetimes.com/design/automotive-design/4007180/Multi-Core-Processors-Driving-the-Evolution-of-Automotive-Electronics-Architectures>. Acesso em: 10 set. 2010.

[6] Entrup, U. L.; Lowne, A. **Choosing a CANbus Industrial Controller Wisely**. Janz Automation Systems GmbH and Saelig Co. Inc. Disponível em: <http://www.janz.de/as/images/stories/Fachartikel/choosing%20a%20canbus%20industrial%20controller.pdf>. Acesso em: 10 set. 2010.

[7] **Application Domains**. Disponível em: <http://www.can-cia.org/index.php?id=30>. Acesso em: 22 set. 2010.

[8] Chen, M. C. **In a Race for Precision? Start your Engines!**. In: CommonWealth Magazine. n. 367, mar. 2007. Disponível em: http://english.cw.com.tw/article.do?action=show&id=3290. Acesso em: 13 out. 2010.

[9] Pfeiffer, O.; Ayre, A.; Keydel, C. **Embedded Networking with CAN and CANopen**. Copperhill Technologies Corporation. 2008. 5 p. Disponível em: < http://copperhillmedia.com/PDF/CAN_and_CANopen_Preview.pdf>. Acesso em: 15 de out. 2010.

[10] Sousa, R. V.; Inamasu, R.Y.; Neto, A. T. **CAN (Controller Area Network): Um Padrão Internacional de Comunicação de Transdutores Inteligentes para Máquinas Agrícolas**. 2001. 2 p. Disponível em : <http://www.cnpdia.embrapa.br/publicacoes/CiT12_2001.pdf>. Acesso em: 10 set. 2010.

[11] **CAN in Automation**. Disponível em: <http://www.can-cia.org/index.php?id=441>. Acesso em: 22 set. 2010.

[12] Leen, G.; Heffernan, D. **Expanding Automotive Electronic Systems**. In-Vehicle Networks. Jan 2002. x p. Disponível em: <http://www.semiconductors.bosch.de/pdf/expanding.pdf>. Acesso em: 22 set. 2010.

[13] Pfeiffer, O. **Selecting CAN controller**. Embedded Systems Academy. Disponível em: <http://www.esacademy.com/en/library/technical-articles-and-documents/can-and-canopen/selecting-a-can-controller.html>. Acesso em: 6 out. 2010.

[14] ARM. **The Architecture for the Digital World**. Disponível em: <http://www.arm.com/products/processors/classic/arm7/arm7tdmi.php>. Acesso em: 21 out. 2010.

[15] LabTools. **Guia do Usuário Ferramenta de Desenvolvimento McBoard LPC2368**. Rev. 3.0. 2010.

[16] NXP. **LPC23xx User Manual**. Rev. 3. 2009. 3 p. Disponível em: <http://www.nxp.com>. Acesso em: 15 out. 2010.

[17] Martin, T. **The Insider's Guide to the NXP LPC2300/2400 Based Microcontrollers.** Hitex (UK) Ltd, 2007. 11 p. ISBN: 0-95499886.

[18] Baumer. **Absolute encoders – bus interfaces**. Disponível em: <http://motion.baumergroup.com/motion/products/downloads/Produkte/PDF/Datenblatt/Drehgeber/PI_BMSH_42_BMMH_42_CANopen_EN.pdf>. Acesso em: 15 out. 2010.

[19] Baumer. **Manual Absolute Encoder with CANopen (bus cover and integrated interface)**. Rev. 1.04. 2009.