Alessandre Rodrigues Geraldo

REDE CAN PARA ACIONAMENTO E CONTROLE DA PLATAFORMA AUTÔNOMA PARA COLETA DE DADOS HIDROLÓGICOS

Trabalho de Conclusão de Curso
apresentado à Escola de Engenharia de
São Carlos, da Universidade de São Paulo

Curso de Engenharia Elétrica com
ênfase em Eletrônica

ORIENTADOR: Prof. Dr. Valentin Obac Roda

São Carlos
2008

Resumo

O fornecimento de água de qualidade às cidades é necessário para que se garanta a saúde das pessoas. E para isso, o monitoramento das variáveis físico-químico da água é fundamental. Os trabalhos de coleta de amostras por técnicos que se deslocam até rios e reservatórios podem ser prejudiciais as suas saúdes devido às adversidades climáticas ou devido à poluição das águas. Uma busca para solução desse problema vem sendo feita pelo Laboratório de Instrumentação Virtual e Microprocessada do Departamento de Engenharia Elétrica/EESC/USP, no desenvolvimento de uma plataforma autônoma para coleta e análise de amostras de água. O presente trabalho tem por finalidade o estudo, a compreensão e a aplicação da rede CAN de comunicação aplicado em um sistema de controle da plataforma autônoma. Foram desenvolvidas unidades de controle microcontroladas distribuídas para utilização na embarcação, sendo que cada uma será responsável pelo acionamento de atuadores e leitura de sensores. Um computador embarcado será utilizado como comando principal do sistema e como interface entre a embarcação e a comunicação com a estação base.

Palavras chave: Microcontroladores, Instrumentação, Veículos autônomos, Redes de instrumentação, Rede CAN

Abstract

The providing of quality water to cities is necessary to ensure the health of people. And consequently, the monitoring of physical and chemical variables of water is crucial. The work of collection of samples by technicians moving up rivers and reservoirs may be harmful to their health due to climatic adversities or due to water pollution. A search for solution of this problem is being made by the Laboratory of Virtual Instrumentation and Microprocessors, Department of Electrical Engineering / EESC / USP, in developing an autonomous platform for collection and analysis of water samples. This paper aims at the study, understanding and application of the CAN network of communication applied to a system of autonomous control of the platform. Have been developed to control microcontroller units distributed for use on the boat, each of which will be responsible for driving actuators and reading sensors. A computer board will be used as the main control system and as an interface between the boat and communication with the base station.

Keywords: Microcontrollers, instrumentation, autonomous vehicles, instrumentation networks, CAN Network

Dedicatória

Aos meus pais Marco Antonio e Isabel pelo incentivo e apoio em meus estudos.

Agradecimentos

Ao Professor Dr. Valentin Obac Roda pela oportunidade da realização deste trabalho.

À amiga Mônica Maria Ramos Germano Travieso pelas dicas de programação e pelos conselhos para a vida.

Lista de abreviaturas e siglas

A/D	*Analog to Digital Converter*
AMP	*Arbitration on Message Priority*
CAN	*Controller Area Network*
CC	Corrente Contínua
CRC	*Cyclic Redundancy Check*
CSMA/CD	*Carrier Sense Multiple Access/ Collision Detection*
DIP	*Dual in Line Package*
EOF	*End of File*
GPS	*Global Positioning System*
I^2C	*Inter Integrated Circuit*
ISO	*International Organization for Standardization*
ISP	*In System Program*
JTAG	*Joint Test Action Group*
kbps	*Kilobits per second*
Mbps	*Megabits per second*
PC	*Personal Computer*
PWM	*Pulse Width Modulation*
RF	*Radiofrequency*
RISC	*Reduced Instruction Set Computer*
RX	*Receive*
SAE	*Society of Automobile Engineers*
SPI	*Serial Peripheral Interface*
TX	*Transmit*
U.C.	Unidade de Controle
UART	*Universal Asynchronous Receiver Transmitter*
USART	*Universal Synchronous and Asynchronous Receiver Transmitter*

Sumário

1. Introdução 2
2. Por que rede CAN? 3
   2.1. CAN - Definição 5
   2.2. Camada Física 5
      2.2.1. Níveis de Tensão 6
      2.2.2. Taxa de transmissão 6
      2.2.3. Máximo comprimento do barramento 7
      2.2.4. O Cabeamento 7
      2.2.5. Terminações 8
   2.3. Camada de Enlace 8
   2.4. Comunicação na rede CAN 8
   2.5. As mensagens CAN 9
   2.6. Tipos de Mensagens 11
   2.7. *Bit Stuffing* 12
   2.8. Detecção de Erros 12
   2.9. Confinamento de Falhas 12
3. Circuitos Ponte H 13
   3.1. O circuito integrado VNH3SP30 14
   3.2. Princípio de operação 15
   3.3. Circuito de teste 17
4. Microcontroladores com Interface CAN 18
   4.1 LPC2129 19
   4.2 PIC18F4580 20
   4.3 AT90CAN128 21
5. Escolha do microcontrolador a ser utilizado 22
6. A arquitetura do AT90CAN128 24
7. Características de programação do microcontrolador AVR 26
8. Construção da placa de controle CAN 28
9. Configuração do sistema de teste 32
10. Programação dos microcontroladores 33
   10.1 Operação do Controlador CAN no AT90CAN128/64 35
   10.2 Configuração e inicialização do controlador CAN 36
   10.3 Filtragem de mensagens CAN 39
   10.4 Mensagens CAN utilizadas 41
   10.5 Comunicação serial no AT90CAN128/64 41
   10.6 Conversor A/D 42
   10.7 Gerador de sinais PWM 42
11. Testes do sistema 43
12. Conclusões 47
13. Anexos 48
   13.1 Programa da U.C. 1 48
   13.2 Programa da U.C. 2 54
14. Referências bibliográficas 58

1. Introdução

A qualidade das águas é um tema de muita importância nos dias de hoje. Principalmente das águas provenientes de rios e represas responsáveis pelo abastecimento de cidades, onde impactos causados pela poluição podem afetar não só a saúde de pessoas como também causar grandes prejuízos econômicos.

Somado ao problema da escassez, torna–se obrigatório a tomada de medidas para que haja um uso sustentável da água. E uma análise química e física da água é fundamental para monitorar sua qualidade.

Antes de ser captada, a água pode ser analisada ainda na represa ou rio, onde técnicos se dirigem através de barcos para que façam a coleta das amostras.

Esse tipo de coleta, ainda que tenha bons resultados, oferecem riscos às pessoas envolvidas, tanto por estar expostas às adversidades climáticas (sol, tempestades) como também no risco de possíveis contaminações devido à poluição das águas.

Na busca de um meio mais seguro e prático para realização da coleta e análise da água, vários trabalhos no laboratório de Instrumentação Virtual e Microprocessada do Departamento de Engenharia Elétrica/EESC/USP vêm sendo desenvolvidos para o desenvolvimento de uma plataforma autônoma para que faça a coleta e análise de amostras de água em reservatórios.

Essa plataforma consiste em um barco de motor de popa, onde foi aplicado um sistema de controle microcontrolado, atuando em motores elétricos para movimentação do volante e do controle de velocidade do barco.

Com o uso de GPS e Bússola eletrônica instalados, o barco pode ser posicionado com precisão, e os dados de controle da plataforma são enviados por uma estação base através de um enlace por RF.

O presente trabalho tem como finalidade a implantação de uma rede de comunicação CAN para controle dos motores e para futura leitura dos instrumentos de posicionamento. Essa rede de comunicação permite tornar o sistema da plataforma flexível a expansões e é um meio mais robusto para seu controle.

Para o controle dos motores, serão utilizados módulos de ponte H semicondutores, permitido através de sinal PWM um controle também da velocidade de giro dos motores.

2. Por que rede CAN?

Atualmente, várias aplicações móveis, como carros, aviões, tratores utilizam a eletrônica embarcada em seus sistemas.

Entende-se por eletrônica embarcada, toda aplicação móvel em que possa ser usada a eletrônica como responsável por algumas tarefas do sistema. Por exemplo, em um automóvel pode-se citar o sistema de alarme, ignição eletrônica, injeção de combustível, painel de instrumentos, controle dos freios, monitoramento de temperatura, etc., todos comandados eletronicamente. [1]

Ao se projetar um sistema de eletrônica embarcada, pode-se optar por concentrar todo controle do sistema em um só local (unidade de controle), aonde chegam todos os sinais de sensores, chaves de comandos, e do mesmo lugar onde partem cabos que irão ser ligados a atuadores do sistema. Isso caracteriza um sistema de arquitetura centralizada como indicado na figura 1. [1]

A princípio, para pequenos sistemas, pode ser mais vantajoso tal arquitetura, mas em um sistema relativamente maior, a arquitetura centralizada apresenta algumas deficiências citadas a seguir:

Primeiramente, o uso de longos cabos desde um sensor ou atuador até a unidade de controle, podem funcionar como antenas captando sinais de fontes externas, causando erros nas leituras desses sinais recebidos.

Em segundo lugar, pode–se citar a dificuldade de expansão deste sistema, sendo que muitas vezes necessite a construção de um circuito superdimensionado para suprir futuras modificações, encarecendo seu valor.

Outra dificuldade dessa arquitetura é a maior complexidade dos circuitos da unidade de controle, e da maior complexidade de programação de um microcontrolador responsável por todo o sistema. Numa futura expansão do sistema, mais linhas de código seriam necessárias e um microcontrolador superdimensionado seria necessário.

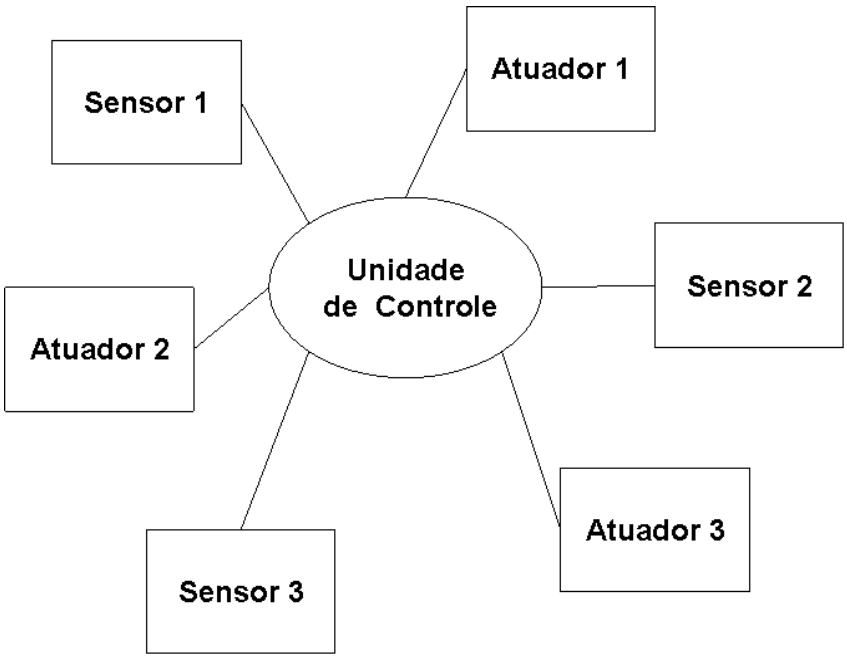

Figura 1 - Exemplo de um sistema de arquitetura centralizada

Outra topologia possível seria a arquitetura distribuída indicada na figura 2. Nessa configuração, cada unidade de controle (U.C.) está localizada próxima aos seus sinais de entrada (sensores) e saída (atuadores). Isso reduz o comprimento de cabos para sinais utilizados, diminuindo assim o efeito de interferências. [1]

Sendo cada unidade de controle responsável agora por um pequeno número de sensores e atuadores, uma menor carga de processamento é exigida dos microcontroladores e também, os programas de cada unidade de controle são menos complexos, pois são responsáveis pelo comando de parte do sistema.

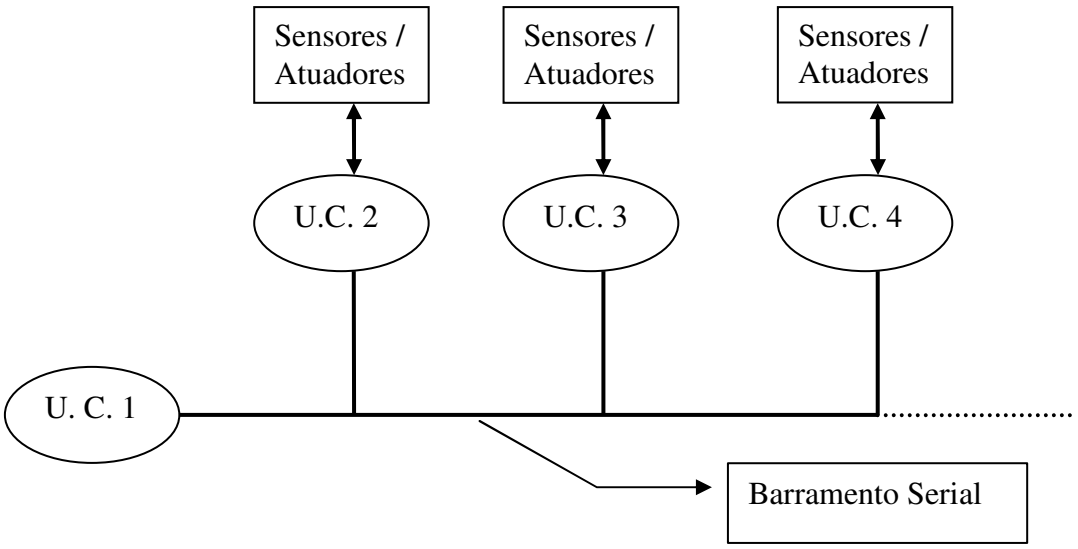

Figura 2 - Exemplo de um sistema de arquitetura distribuída

Outra vantagem dessa topologia é sua capacidade de expansão, bastando interligar à rede do sistema novas unidades de controle que por sua vez se conectará a outros sensores e atuadores.

Porém, para que haja o correto funcionamento desse sistema, é necessário um meio de comunicação rápido e confiável que interligue todas as unidades de controle, para que o sistema opere de forma robusta.

Buscando obter uma proteção contra interferências e ao mesmo tempo um meio simples de interconexão, a comunicação entre as unidades deve ser na forma serial, e o cabo utilizado é um par trançado de fios, eficiente na minimização de interferências eletromagnéticas.

Além das características citadas, a comunicação serial deve atender aos seguintes requisitos:

a) Alta taxa de transmissão de dados entre as unidades;
b) Facilidade de expansão, bastando conectar um novo circuito ao barramento de comunicação;
c) Baixo índice de erros dos dados transmitidos/recebidos;
d) Robustez do sistema a falhas que venham ocorrer em algumas das unidades de controle, na qual não deve afetar o funcionamento de todo o sistema;

Na busca destas características, um barramento de comunicação serial foi desenvolvido pela BOSCH nos anos 80, o CAN (*Controller Area Network*), que teve propósito original, atender as especificações de sistemas de eletrônica embarcada em automóveis. [2]

2.1. CAN - Definição

CAN (Controller Area Network) é um sistema de comunicação serial, especialmente desenvolvido para interligação de dispositivos "inteligentes" bem como sensores e atuadores em um sistema ou subsistema. [3]

É um protocolo de comunicação serial síncrono, com característica multimestre, i.e., qualquer unidade da rede pode ora ser mestre, comandando a rede, ou escrava, respondendo a comandos de outra unidade.

O protocolo CAN tem sua estrutura dividida em duas partes principais assim denominadas: camada física e camada de enlace, que são discutidas a seguir.

2.2. Camada Física

A Camada Física define o modo de transmissão dos sinais entre diferentes unidades a respeito das propriedades elétricas. [4]

Existem formas diferentes de se interconectar unidades em uma rede CAN, a saber: [5]

a) O tipo mais usado é definido pela norma ISO 11898-2, que consiste no uso de um par de fios trançados, onde o sinal é transmitido de forma diferencial, atenuando os efeitos de interferências eletromagnéticas. Este padrão é conhecido também como CAN de alta velocidade;
b) Outra norma, a ISO 11898-3, define outro esquema também utilizando de um par de fios balanceados, porém a uma taxa de comunicação menor. Essa topologia tem a vantagem de ser mais segura à falhas, pois pode funcionar mesmo se um dos fios for aberto ou curto-circuitado à alimentação (+V ou –V);
c) Outro padrão (SAE J2411) define a comunicação não diferencial (um fio comum mais o de dados);

Para o trabalho, optou-se pelo uso da transmissão por dois fios em alta velocidade.

2.2.1. Níveis de Tensão

No caso da transmissão por dois fios, estes são identificados como CAN_H e CAN_L. O nível de tensão entre essas linhas determina o estado lógico do barramento.

Em CAN, diferente dos bits "1" e bit "0" normalmente utilizados em aplicações lógicas, é utilizado as denominações de bit recessivos e bit dominantes.

Um bit recessivo se obtém quando a diferença de tensão entre CAN_H e CAN_L é de zero volt, isto é obtido quando os dois fios do barramento estão a um nível de 2,5V. Já o bit dominante, é estabelecido quando há uma diferença de 2 volts entre esses terminais, sendo CAN_H em 3,5V e CAN_L em 1,5V (figura 3). [6]

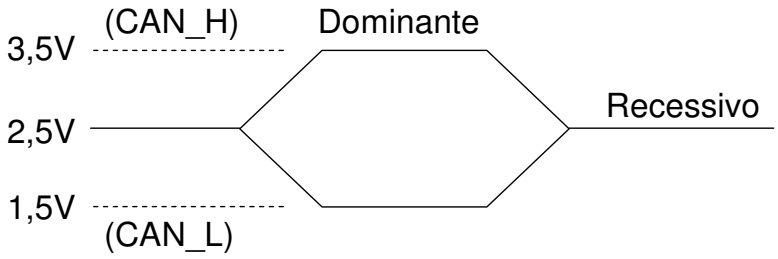

Figura 3 - Níveis de tensão no Barramento CAN

2.2.2. Taxa de transmissão

Para CAN de alta velocidade, a norma define em 1Mbit/s a máxima taxa de transmissão. Já para CAN de baixa velocidade a máxima taxa é de 125kbit/s, e para CAN de um fio a taxa é em torno de 50kbit/s. [1]

2.2.3. Máximo comprimento do barramento

Para taxa máxima de 1Mbps, é admitido um comprimento máximo do cabo de até 40 metros. Este comprimento máximo é obtido levando em consideração o tempo de amostragem de um bit na rede, sendo que um bit enviado deve percorrer toda a rede e então retornar ao ponto de origem para só então ser amostrado. [5]

Conclui-se que a velocidade de um sinal pelo cabo é um limitante para seu comprimento.

Abaixo estão outros máximos comprimentos por taxa de transmissão:

- 100 metros em 500 kbps;
- 200 metros em 250 kbps;
- 500 metros em 125 kbps;
- 6 km em 10 kbps;

O gráfico abaixo (figura 4) ilustra o comprimento do cabo por taxa de transmissão:

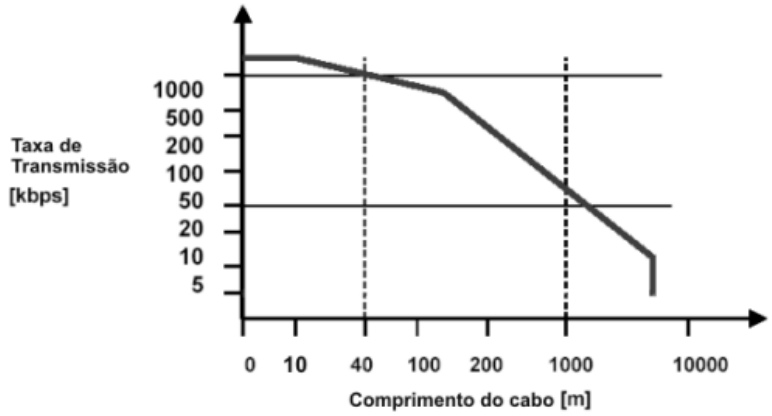

Figura 4 - Comprimento do cabo x taxa de transmissão [6]

2.2.4. O Cabeamento

A norma ISO 11898 determina que a impedância do cabo do barramento CAN deve ser de 120 ohms, porém impedâncias entre 108 e 132 ohms são permitidas. Esse cabo é constituído de um par de fios trançados que também podem ser blindados. [5]

2.2.5. Terminações

Para não haver reflexões de sinal nas extremidades do barramento CAN, e garantir os níveis de tensão adequados, são utilizados resistores de 120 ohms nas extremidades do barramento.

2.3. Camada de Enlace

Na camada de Enlace, o protocolo CAN se divide em duas subcamadas: Camada de objeto e Camada de transferência. [4]

O escopo da Camada de objeto é:

- Encontrar quais mensagens serão transmitidas;
- Decidir quais mensagens recebidas pela camada de transferência serão de fato usadas;
- Prover uma interface à camada de aplicação;

Já o escopo da Camada de transferência constitui o protocolo de transferência, o que envolve:

- Controle de pacotes de dados;
- Realizar arbitração;
- Checagem de erro;
- Sinalização de erro;
- Confinamento de Falhas;

2.4. Comunicação na rede CAN

O protocolo CAN utiliza o conceito de comunicação chamado *CSMA/CD and AMP*. [6] Nesse conceito de comunicação, quando uma unidade da rede quer transmitir seus dados, primeiramente ela verifica se o barramento esta livre para ser utilizado, e isso é checado analisando os níveis de tensão da rede.

Caso esteja livre, e ao mesmo momento, duas ou mais unidades tentam transmitir seus dados, entra em atuação o sistema de arbitração não destrutiva que fará com que a mensagem de maior prioridade seja a transmitida. Isso é explicado a seguir.

Como já mencionado, a comunicação na rede CAN é feita de forma serial, com os bits denominados recessivos e dominantes. Os nomes sugestivos, recessivo e dominante estão diretamente ligados ao modo de arbitração não destrutiva da rede CAN.

Quando duas ou mais unidades enviam um bit ao barramento CAN, todas as unidades checam se este bit está na rede de fato. Aquela unidade que enviou um bit dominante no barramento irá se sobrepor a todas as outras que enviarem um bit recessivo, fazendo-as pararem

de se comunicar na rede. Com isso, aquela unidade que enviou mais bits dominantes, terá sua transmissão concluída sem a interferência de outros que estavam querendo também se comunicar. Dessa maneira, há como dar mais prioridade a uma mensagem que a outra.

A figura 5 exemplifica a arbitração não destrutiva CAN para o caso de três unidades transmitindo ao mesmo tempo na rede:

Os níveis baixos representam os níveis dominantes na comunicação:

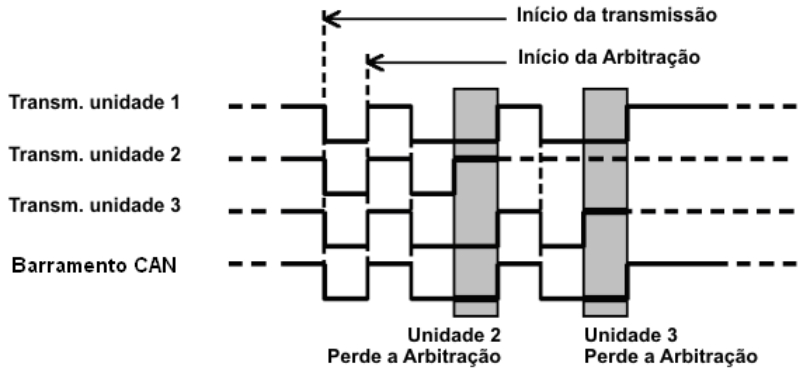

Figura 5 - Exemplo da arbitração por prioridades de mensagens na rede CAN [6]

Nesse exemplo, a unidade 1 vence a arbitração, portanto sua informação é a que será transmitida na rede CAN.

A comunicação na rede CAN é feita de forma síncrona, sendo que o sincronismo de todas as unidades da rede é estabelecido no início de cada mensagem enviada à rede. Ocorre novamente uma sincronização em todas as transições de bits recessivos para dominantes.

A rede CAN utiliza-se da transmissão de dados "*multicast*", i.e., toda mensagem enviada na rede é "ouvida" por todos dispositivos interligados à rede. Portanto, não há como enviar uma mensagem de uma específica unidade para outra em particular. Fica, portanto, como tarefa de cada unidade, filtrar aquelas mensagens que são de fato úteis a ela.[5]

2.5. As mensagens CAN

As informações sobre mensagens CAN e seus tipos foram obtidos nos materiais da especificação CAN [4]

Entende-se por mensagem, um conjunto de dados enviados/recebidos entre unidades em uma rede CAN. Atualmente existem duas versões de mensagens CAN que podem ser utilizadas. A versão 2.0A e 2.0B, as quais são mostradas na figura 6: [4], [6]

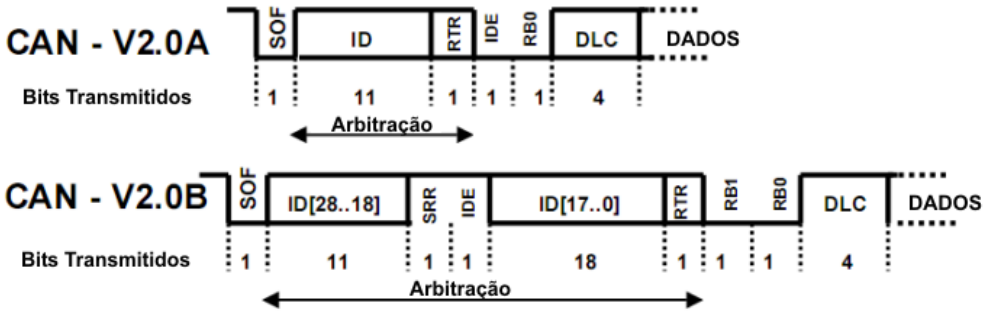

Figura 6 - Formatos de mensagens CAN nas versões 2.0A e 2.0B

Na versão 2.0A são utilizados 11 bits identificadores da mensagem, i.e., podem ser transmitidas 2^{11} tipos mensagens diferentes na rede. Já na versão 2.0B, 29 bits são usados para identificação das mensagens, expandindo agora para 2^{29} mensagens possíveis na rede.

O bit IDE é utilizado para identificar entre quais versões a mensagem está sendo enviada, sendo dominante para versão 2.0A.

No caso da mensagem CAN 2.0B o bit RTR (*Remote transmission Request*) muda de posição deixando em seu lugar um bit recessivo SRR *(Substitute Remote Request*). Os bits RB0/RB1 são bits reservados, sem função na mensagem. O campo DLC (*data length code*) indica o numero de bytes que serão transmitidos no campo DADOS.

A segunda parte da mensagem é comum em ambas as versões (figura 7).

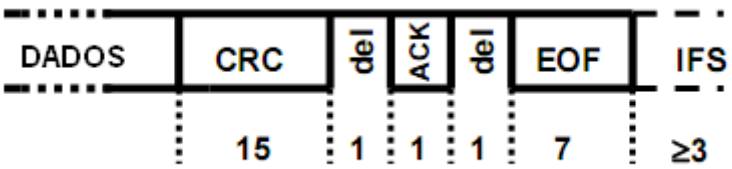

Figura 7 - Parte final da mensagem CAN

No campo CRC estão 15 bits gerados pelo transmissor e que são recalculados no receptor para verificação de algum possível erro dos bits até então enviados. Entre os bits delimitadores (del) está o bit ACK que é recessivo pra o transmissor da mensagem e dominante no receptor, caso o receptor reconheça como válida a mensagem. Essa informação é então verificada pelo transmissor.

A mensagem é então finalizada no campo EOF (*End of File*) com 7 bits recessivos seguidos de 3 bits recessivos IFS (*Interframe space*) usado como um separador para a próxima mensagem que possa ser enviada.

As mensagens enviadas em uma rede CAN não possuem em seu corpo nenhum endereço indicando para quem interessa a mensagem, fica como tarefa da unidade receptora reconhecer a mensagem pela informação por ela carregada.

2.6. Tipos de Mensagens

Podemos dividir as mensagens CAN em 4 categorias de pacotes de dados:

1. *Data Frame*: esta mensagem leva informação no campo dados do transmissor para os receptores: caracterizada pelo bit RTR=0;
2. *Remote Frame*: Esta mensagem é feita pelo transmissor quando o mesmo requisita dados (Data Frame) de outra unidade na rede. Caracterizada com bit RTR=1 e o campo DADOS não existe.
3. *Error Frame* – mensagem de erro – Caso uma unidade da rede detecte a falha durante a transmissão de uma mensagem, um pacote de erro é então gerado por este, informando ao transmissor a necessidade de retransmitir a informação (figura 8).

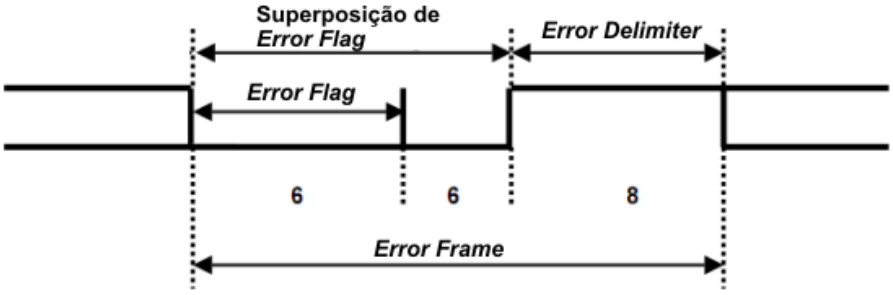

Figura 8 - Mensagem de erro

Os 6 primeiros bits dominantes (*Error Flag*) são enviados pela unidade que detectou o erro, fugindo da regra do *bit-stuffing* e disparando mensagens de erro de outros nós da rede. Por isso os próximos 6 bits são reservados para que outras unidades enviem suas mensagens de erro. O campo *Error delimiter* é formados de bits recessivos, que quando são detectados pelo seu transmissor, indica que a rede está pronta para o envio de uma mensagem.

4. *Overload Frame* – Essa mensagem é enviado por uma unidade na rede que esteja com sobrecarga de armazenamento de dados (figura 9). Assim, esse pacote de dados faz com que os transmissores aguardem até o envio de outras mensagens. Podem ser enviados até dois *overload frame* consecutivos

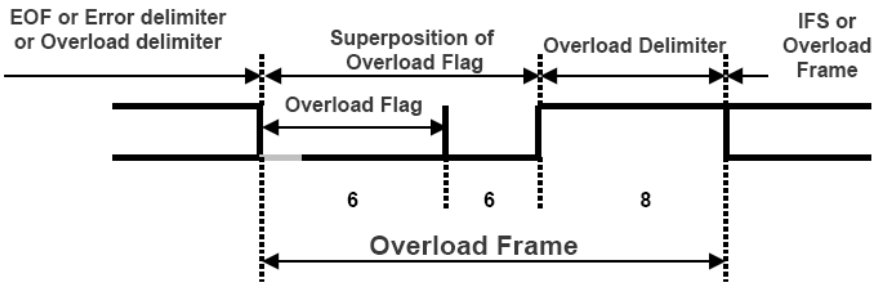

Figura 9 - Mensagem de *overload*

O *overload flag* foge da regra de intermissão (dentro do *interframe*), fazendo com que todas as unidades também gerem a mesma sinalização de *overload.* A faixa *overload delimiter* é igual à faixa *error delimiter* do *error frame*

2.7. *Bit Stuffing*

Além do CRC e ACK no corpo da mensagem, outro item que se soma à segurança na transmissão de dados via CAN é o *bit stuffing*, realizado da seguinte forma:

Quando se transmite uma seqüência de 5 bits repetidos, o próximo deve ser um bit de valor complementar, indicando que a rede está em operação normal [4]. No receptor, este bit é então retirado para remontar a mensagem. Por essa característica, o tamanho de uma mensagem CAN varia de um pacote pra outro pacote.

O *bit stuffing* somente ocorre em uma parte da mensagem CAN situado desde o início da transmissão (SOF - *Start of Frame*) até o final do campo CRC.

2.8. Detecção de Erros

O que leva o protocolo CAN ser altamente confiável é seu numero de mecanismos de detecção de erros que atuam simultaneamente em uma transmissão:

a) Erro de Bit: esta checagem é feita fora do campo de arbitração e do campo de reconhecimento (ACK), onde cada bit enviado a rede é checado pelo seu transmissor.
b) Erro de *Bit-stuff*: caso não seja detectado o *bit-stuff* no campo adequado.
c) Erro de CRC: caso o receptor calcule como errado os bits CRC gerados pelo transmissor.
d) Erro de Forma: este erro é gerado quando em campos de formato fixo, como EOF, Interframe Space, delimitadores do ACK ou delimitadores do CRC possuírem bits dominantes causando violação de forma.
e) Erro de ACK: quando o sinal dominante de ACK do receptor não chega ao transmissor.

2.9. Confinamento de Falhas

Juntamente com a detecção de erros, outro recurso que garante a confiabilidade da rede é o confinamento de falhas. Pode ocorrer que certas unidades da rede não estejam operando de forma correta. Por esse motivo as unidades podem estar em 3 estados possíveis, de acordo com a contagem de dois registradores internos aos controladores CAN de cada unidade, sendo um para contagem de erros de transmissão (TEC - *Transmit Error Counter*) e outro para contagem de erros de recepção (REC - *Receive Error Counter*). Um erro em ambos os casos faz o contador correspondente aumentar sua contagem em oito. Já um acerto, diminui em contador correspondente em uma unidade. [6]

De acordo com os valores desses contadores, as unidades numa rede CAN podem estar em três possíveis estados:

a) "*Error Active*" – Unidades operando neste estado estão operando em seu estado normal, onde podem atuar ativamente na rede, inclusive podendo enviar pacote de erro com o conjunto de bits "*error flag*" dominantes, informando à rede a ocorrência de falhas. Os contadores TEC e REC dessas unidades estão abaixo de 128;

b) "*Error Passive*" – Caso um dos contadores TEC ou REC tenham valores maior ou igual a 128, a unidade em questão pode ainda enviar e receber dados, porém somente pode enviar pacotes de erro com os bits "*error flag*" recessivos, que só serão atendidos se esta for a única unidade transmitindo num momento.

c) "*Bus Off*" – Caso o contador TEC exceda 255, a unidade em questão não pode mais transmitir ou receber dados da rede. Uma unidade neste estado só poderá voltar a comunicar caso o sistema seja reiniciado.

3. Circuitos Ponte H

Uma das metas do trabalho é a utilização de um controle com pontes H semicondutoras para o acionamento de cada um dos motores utilizados na embarcação, sendo um para o controle de direção e outro para o controle do sentido e aceleração do motor do barco (figura 10).

Devido à característica distribuída do sistema, cada atuador será instalado próximo ao motor. Isso permite com que a alimentação dos módulos seja feita de forma simplificada como um barramento distribuindo a alimentação a todos os módulos, facilitando nas possíveis expansões (figura 11).

Figura 10 - Detalhe dos comandos do barco

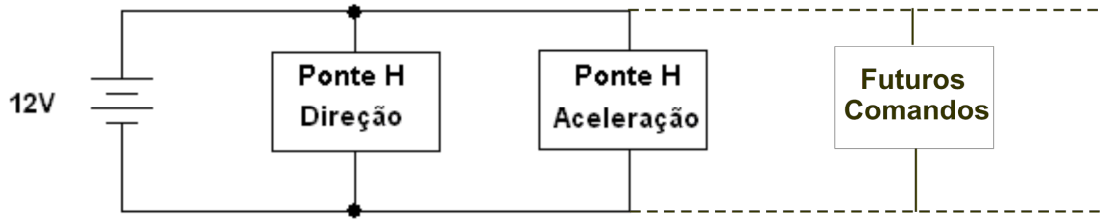

Figura 11 - Alimentação dos controles em um único barramento

O circuito atuador permite fazer girar um motor em ambos os sentidos de rotação através de um comando eletrônico.

Por ser constituídas de circuitos semicondutores que suportam trabalhar em regime de freqüência de vários kilohertz, pode-se controlar a velocidade de cada motor via modulação de largura de pulso (PWM). Este sinal PWM pode ser gerado pelo microcontrolador.

3.1. O circuito integrado VNH3SP30

Em busca de ocupar menor espaço para o circuito, optou-se no uso de um circuito ponte H integrado, onde em um único componente se tem a parte de controle e de potência.

Este componente (figura 12 e figura 13), poder ser operado com tensão máxima de 40 V e suportar correntes de até 30 A. Possui internamente circuito de proteção contra sobretemperatura, sobretensão, subtensão. Pode operar em modulação de largura de pulso com freqüências de até 10kHz. [7]

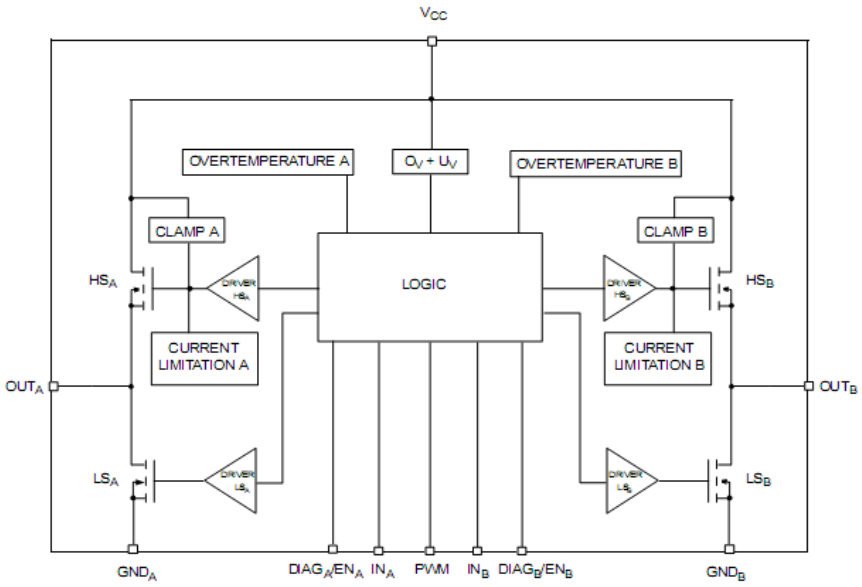

Figura 12 - Diagrama em blocos do integrado VNH3SP30 [7]

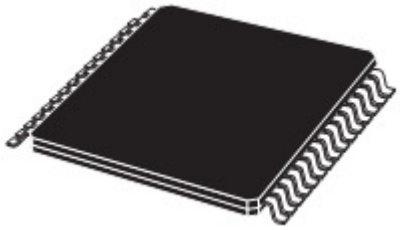

Figura 13 - Encapsulamento do integrado VNH3SP30 [7]

3.2. Princípio de operação

Este circuito tem seu estágio de entrada que opera com níveis de tensão de 5 volts, portando facilitando o interfaceamento com o microcontrolador.

Na figura 14 está um circuito proposto pela ST para operação do componente:

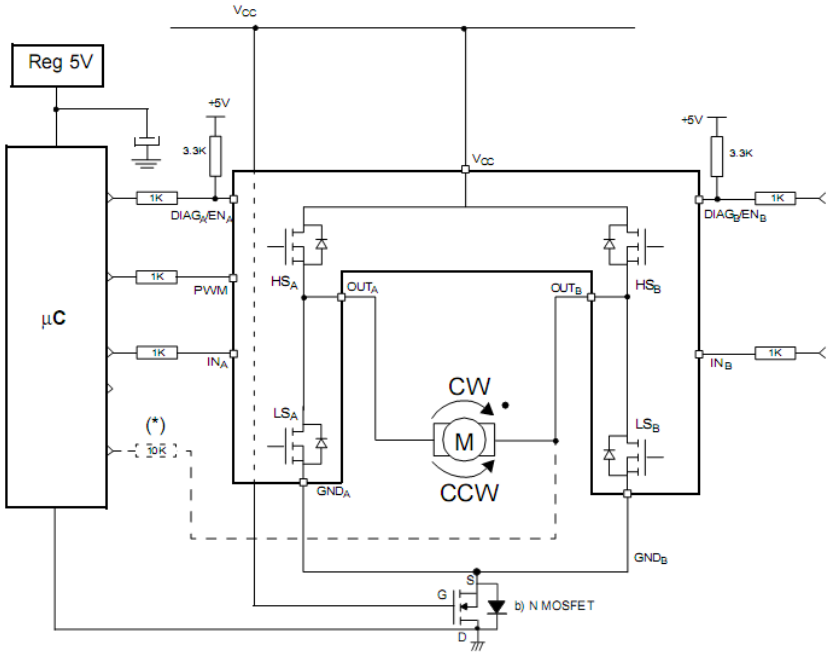

Figura 14 - Circuito proposto pelo fabricante ST [7]

O motor é ligado aos terminais OUT_A e OUT_B. Os pinos $Diag_A/EN_A$ e $Diag_B/EN_B$ atuam como entrada de habilitação e são mantidos em nível lógico 1 para ativar os dois braços da ponte H. O controle de direção do motor é feito através dos pinos IN_A e IN_B, sendo que para cada combinação, um par de transistores entrará em saturação, permitindo a passagem de corrente por eles obedecendo a ordem indicada a seguir (tabela 1):

Tabela 1 - Estados de operação do integrado VNH3SP30.

IN_A	IN_B	Transistores ativos	OUT_A	OUT_B	Estado do Motor
1	1	HS_A, HS_B	VCC	VCC	Parado em VCC
1	0	HS_A, LS_B	VCC	GND	Girando sentido Horário[1]
0	1	HS_B, LS_A	GND	VCC	Girando sentido anti-horário[1]
0	0	LS_A, LS_B	GND	GND	Parado em GND

O transistor MOSFET canal N na parte de baixo do circuito, é usado como proteção de inversão de polaridade da bateria, fazendo com que não circule corrente reversa pelo circuito.

O resistor de 10K é uma sugestão opcional para detecção pelo microcontrolador de carga aberta.

Em caso de falhas, como curto circuito da carga para VCC ou GND, um circuito de proteção interna atua, e essa falha pode ser monitorada através dos pinos $Diag_A/EN_A$ e $Diag_B/EN_B$ atuando agora como saída (Diag – *diagnostic*), sendo que um ou outro irá ter nível lógico 0.

O circuito possibilita comandar o motor para girar em ambas as direções ou mantê-lo parado quando não há diferença de potencial entre seus terminais.

Ficará a cargo do microcontrolador, gerar sinais PWM para o controle de velocidade/potência do motor. Esses sinais são ondas retangulares de freqüência constante, porém variam a proporção de nível alto e baixo, chamado ciclo de trabalho. Em 0% deste ciclo, a tensão no motor é de zero volt. Já em 100% de ciclo de trabalho, o nível de tensão no motor é o máximo possível e constante.

[1] O motor somente irá girar caso o pino PWM estiver em nível lógico 1. Em nível 0, este pino faz com que os transistores LS_A e LS_B fiquem no estado de corte, não conduzindo corrente.

Na figura 15 é mostrada em forma de gráfico, exemplos de sinais PWM com ciclos de trabalho diferentes:

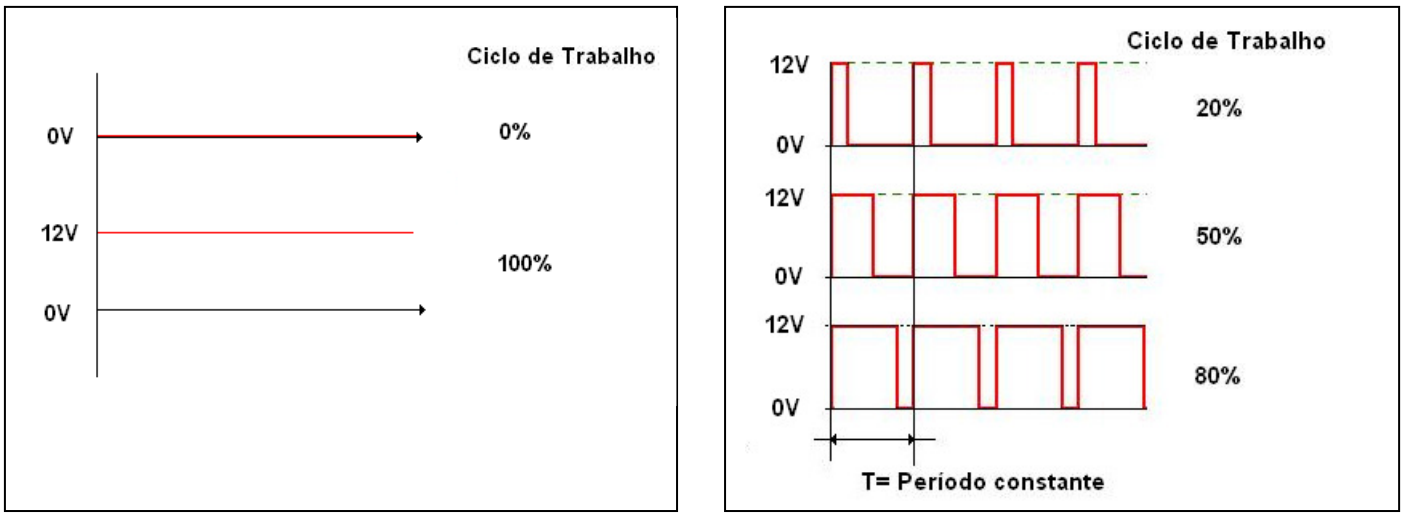

Figura 15 - Formas de onda PWM

Com freqüência apropriada, esses sinais serão vistos pelo motor, devido sua indutância característica, como um nível médio de tensão DC proporcional ao ciclo de trabalho do sinal PWM, sendo que serão filtradas componentes de freqüência maior ou igual à fundamental.

3.3. Circuito de teste

Um circuito foi montado para o teste do componente VNH3SP30 (figura 16), ligado a um motor DC. Este circuito é semelhante ao proposto pelo fabricante (figura 14).

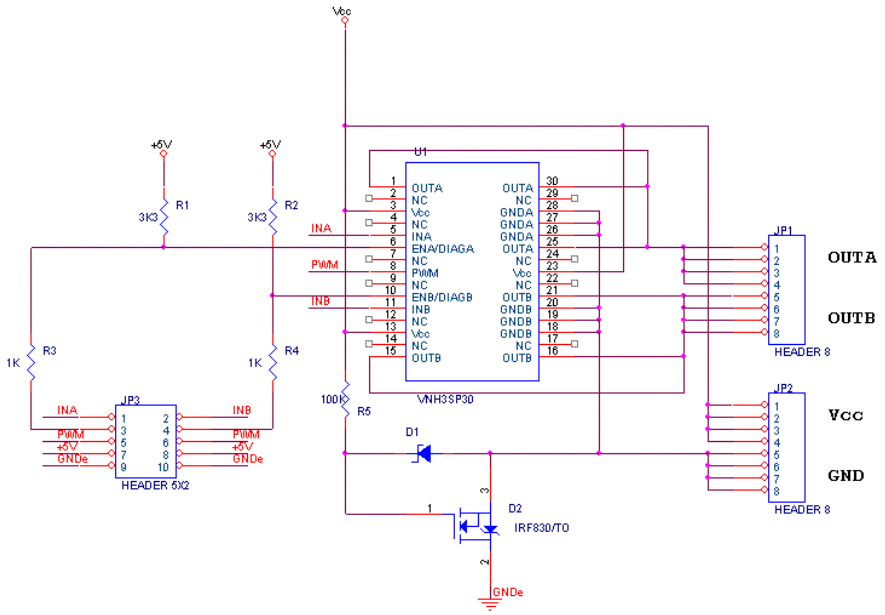

Figura 16 - Diagrama do acionador do motor de corrente contínua

Aplicando-se um sinal PWM e com as combinações de comandos da tabela 1 foi verificado o funcionamento do motor para ambos os sentidos de rotação e controlando sua velocidade.

4. Microcontroladores com Interface CAN

Para interface de cada conjunto de sensores e atuadores à rede CAN, são utilizadas as unidades de controle (U.C.).

Cada U.C. consiste em três componentes principais: um microcontrolador, um controlador CAN e um transceptor CAN (figura 17).

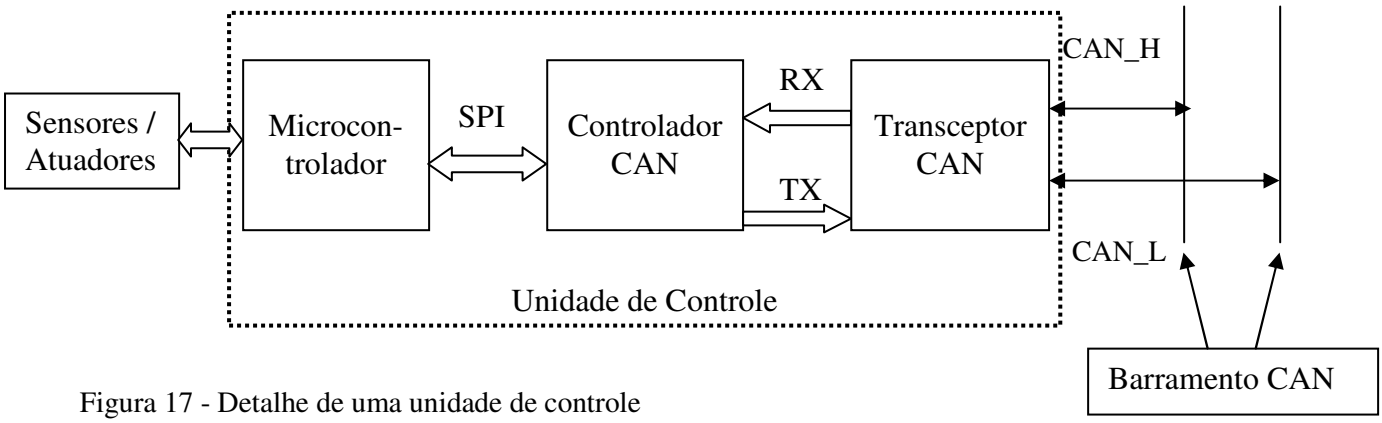

Figura 17 - Detalhe de uma unidade de controle

O transceptor tem a função de combinar os sinais de transmissão e recepção do controlador CAN e adaptá-los aos níveis de tensão diferencial necessários na comunicação do barramento CAN.

Devido a complexidade da realização em *software* de um protocolo CAN, além do custo de processamento que esse protocolo exigiria do microcontrolador, foram desenvolvidos os controladores CAN, que são circuitos integrados dedicados à processar, decodificar, filtrar, receber e transmitir mensagens no protocolo CAN.

Esse controlador normalmente utiliza-se de um barramento serial síncrono (SPI) para se comunicar com o microcontrolador. Para tanto deve ser feita toda programação no microcontrolador para se ter acesso a esse componente.

Outra opção, mais vantajosa atualmente, é a utilização dos microcontroladores que já possuam esse controlador CAN incorporado, tornando mais simples sua programação.

Procurou-se dentre os microcontroladores disponíveis atualmente, aquele que melhor cumpre com as tarefas requeridas no projeto, aliado ao preço, à disponibilidade de ferramentas de desenvolvimento como *softwares* de programação, interfaces para gravação e testes.

Foram examinadas três plataformas diferentes de microcontroladores com interface CAN incorporado: O LPC2129 da NXP, o PIC18F4580 da Microchip e o AT90CAN128 da Atmel. A seguir, são listadas as características principais de cada um desses microcontroladores.

4.1 LPC2129

Este é um microcontrolador com plataforma ARM7 de 32 bits produzido pela NXP. Construído em um encapsulamento SOT314-2 de 64 contatos. Este dispositivo conta com duas interfaces CAN 2.0B, possui memória de programa de 256 kbytes, conversor A/D de 10 bits, 2 portas seriais UART, 1 porta serial I^2C, e duas seriais síncronas SPI. Além disso, possui dois temporizadores de 32bits, com recurso para utilizar até seis saídas PWM. Pode ser programado no circuito. [8]

O fabricante Keil possui uma placa de desenvolvimento (figura 18) para este microcontrolador, onde se encontra montado junto com componentes de interfaceamento CAN e serial, e conectores que dão acesso a essas portas, leds indicadores, um potenciômetro para o acesso do conversor A/D e uma Interface JTAG especial para simulação e programação do o microcontrolador no circuito.

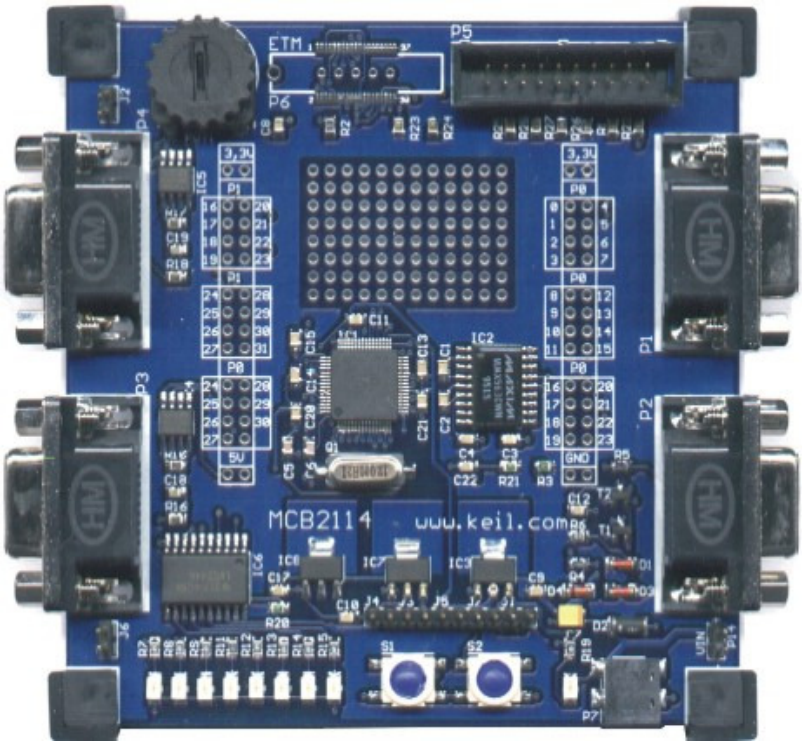

Figura 18 - Plataforma de desenvolvimento para o Microcontrolador LPC2129

Junto à placa, acompanha *software* de avaliação desenvolvido pela própria Keil para programação em linguagem C, com limite de tamanho dos programas de até 16 kbytes. Este kit foi cotado em 109 dólares. [2]

[2] Valor obtido no site www.lpctools.com. Acesso em outubro/2008.

4.2 PIC18F4580

Este microcontrolador de 8 bits produzido pela Microchip pode ser encontrado em encapsulamento DIP de 44 pinos. Conta com uma interface CAN 2.0B, 32 kbytes de memória de programa, conversor A/D de 10 bits, possui porta de comunicação serial USART, além de comunicação serial síncrona SPI e I^2C. Possui 3 temporizadores de 16 bits e pode ser utilizado até 4 saídas PWM. Conta também com recurso de programação em circuito. [9]

Para esse microcontrolador, foi encontrada uma placa de desenvolvimento do fabricante CCS (figura 19), composta de além deste, mais três unidades de controle, sendo uma com o PIC16F876A que utiliza de um controlador CAN externo MCP2515, mais duas expansões CAN MCP25050, que são componentes pré-programados de fábrica que respondem a específicas mensagens CAN. Acompanha compilador para programação em linguagem C, além do programador em circuito. A Microchip disponibiliza *software* de programação gratuito somente para linguagem assembly.

Figura 19 - Plataforma de desenvolvimento para o Microcontrolador PIC

Foi cotado em 549 dólares.[3]

[3] Valor obtido no site www.ccsinfo.com. Acesso em outubro/2008.

4.3 AT90CAN128

Fabricado pela Atmel, este é um microcontrolador da linha AVR de tecnologia RISC de 8 bits pode ser obtido em encapsulamento TQFP de 64 pinos. Possui 128 Kbytes de memória de programa, um controlador CAN configurável entre 2.0A e 2.0B, interface JTAG, duas portas seriais USART, uma porta serial SPI, uma I^2C, dois temporizadores de 16bits, conversor A/D de 10 bits. Permite também programação em circuito. [10]

A plataforma de desenvolvimento é produzida pela própria Atmel (figura 20), possui conectores para acessar portas seriais, CAN, I^2C, conexão para com o programador.

Possui alguns componentes de interface com o usuário, como, cinco teclas, oito LEDS, sensor de temperatura, sensor de luminosidade, e um alto-falante.

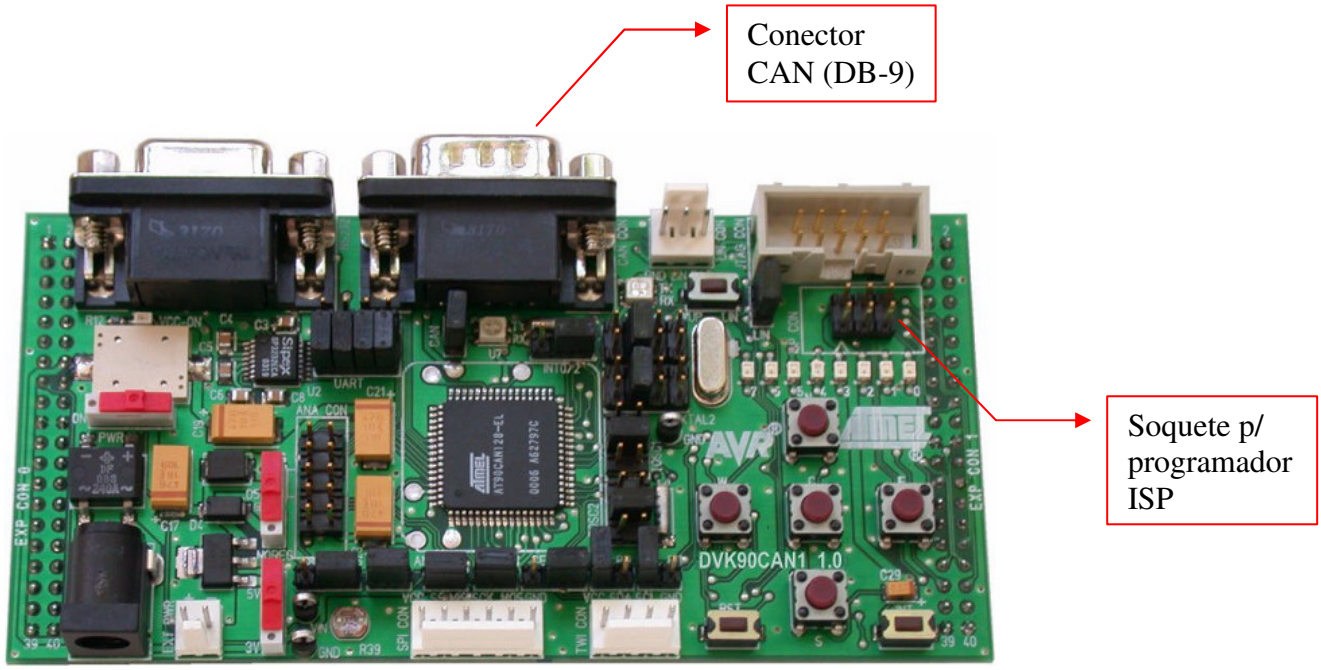

Figura 20 - Placa de desenvolvimento para o microcontrolador AT90CAN128

Acompanha o Kit, dois CDs incluindo documentação, exemplos de códigos de programas, mais *softwares* gratuitos para programação em assembly ou linguagem C.

Foi cotado em 119 dólares. [4]

[4] Valor obtido no site www.digikey.com. Acesso em outubro/2008.

5. Escolha do microcontrolador a ser utilizado

No caso do LPC2129 foram verificadas grandes qualidades em sua capacidade de processamento, além dos recursos de conversores A/D e saídas PWM. Outro ponto é seu preço equiparável aos microcontroladores de 8-bits. Um único ponto que dificultaria a realização com esse componente é na construção de circuito com esse componente, que possui terminais muito próximos um do outro, dificultando na hora da soldagem.

Já com o PIC18F4580 não temos a dificuldade de soldagem, devido ao seu encapsulamento ser do tipo DIP. Possui recursos de interfaces semelhantes aos outros micros, porém o maior custo no uso deste microcontrolador está na compra de *softwares* de programação em linguagem C e em dispositivos dedicados para programação do PIC.

Enfim o AT90CAN128 possui recursos equivalentes aos outros dois microcontroladores, além de possuir encapsulamento conveniente para construção de circuitos no laboratório, e da disponibilidade de *softwares* de programação gratuitos tanto para linguagem assembly quanto em linguagem C. Outra vantagem para o uso deste microcontrolador, é que foi construído no laboratório um programador ISP para a família AVR de microcontroladores (figura 21, figura 22), com um circuito que utiliza da porta paralela do computador, não necessitando componentes dedicados. Este circuito permite a programação do microcontrolador na própria placa, bastando ligar um cabo desta até a placa de desenvolvimento (figura 20). Levando em consideração os fatores para a escolha do microcontrolador, ficou decidido no uso do componente AT90CAN128 da Atmel para ser utilizado no projeto da implantação das unidades de controle da rede CAN.

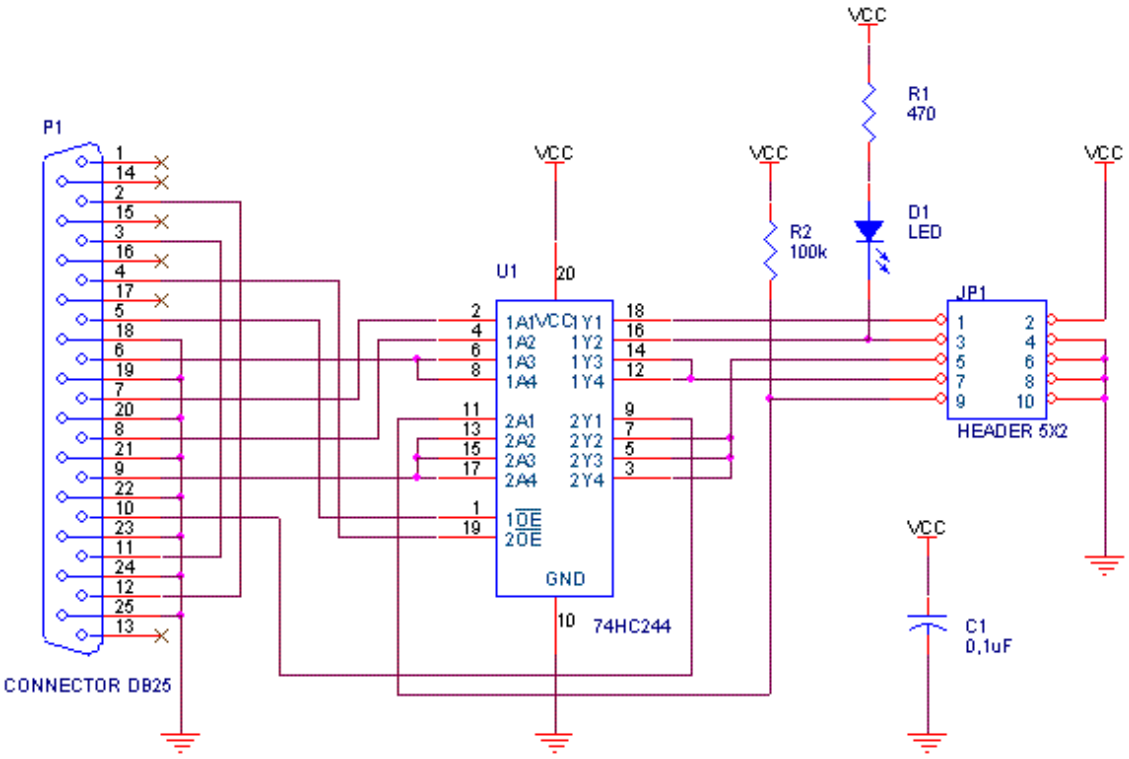

Figura 21 - Diagrama do circuito do programador ISP para AVR

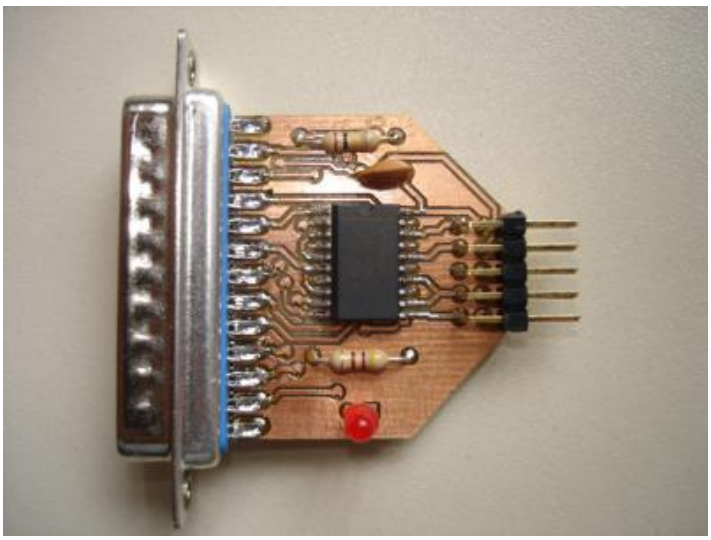

Figura 22 - Circuito final do programador ISP Para microcontroladores AVR

O *software* para utilizar o programador é também obtido gratuitamente na internet.[5]

[5] Software incluído no conjunto de programas WinAVR. Disponível em http://winavr.sourceforge.net

6. A arquitetura do AT90CAN128

Na figura 23 está o diagrama em blocos do microcontrolador AT90CAN128

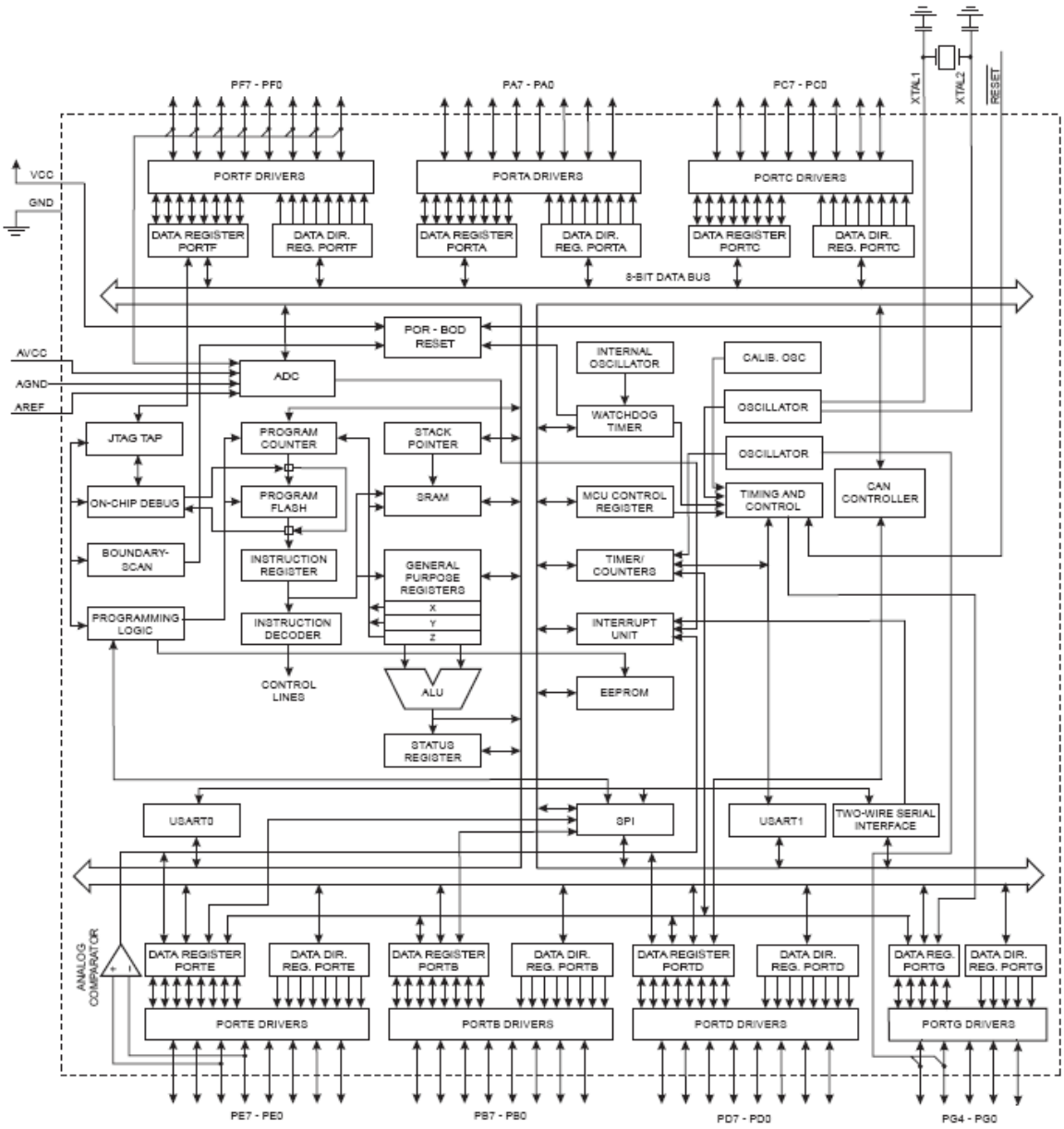

Figura 23 - Diagrama em blocos do microcontrolador AT90CAN128 [10]

Já foram mencionados anteriormente os principais recursos deste microcontrolador, porém pela escolha deste componente, traremos mais detalhes de suas características:

a) Microcontrolador de 8 bits, de alta performance e de baixo consumo: à freqüência de 16MHz, é capaz de realizar 16 milhões de instruções por segundo. Em modo ativo de operação, com freqüência de clock de 16MHz e alimentado em 5V, consome uma corrente de 37mA. Pode também operar em 3V de alimentação com freqüência de clock máxima de 8MHz, onde sua corrente de consumo cai para 10.5mA;

b) Possui conjunto reduzido de instruções: utilizando da arquitetura RISC é capaz de processar 133 instruções distintas;

c) 32 registradores de 8 bits de uso geral além dos registradores dos periféricos internos;

d) Memórias não voláteis para programação e armazenamento de dados: no caso do AT90CAN128, são 128kbytes de memória de programa e 4kbytes de memória de dados. Essas duas memórias são reprogramáveis;

e) 4kbytes de memória estática (volátil) para dados;

f) Capacidade de emulação via porta JTAG: por esta porta, pode-se também fazer a programação em circuito da memória Flash;

g) Capacidade de programação da memória Flash em circuito via portas seriais (UART, CAN);

h) Controlador CAN 2.0A e 2.0B: o controlador CAN deste microcontrolador pode receber e processar mensagens tanto no formato 2.0A como também na versão 2.0B;

i) Possui Watchdog configurável: pode ser usado para evitar que o micro entre em laço infinito;

j) 4 temporizadores/contadores: sendo dois de 8 bits e outros dois de 16 bits: utilizando com comparadores internos, pode ser usados na geração de sinais modulados em largura de pulso (PWM);

k) Conversor analógico/digital de 10 bits: pode ser compartilhado para 8 canais distintos;

l) Comparador analógico;

m) Interface serial de 2 fios;

n) 2 portas seriais assíncronas USART;

o) 1 interface serial síncrona SPI;

7. Características de programação do microcontrolador AVR

Quando foram desenvolvidos, os microcontroladores AVR tiveram suas características internas determinadas para melhor eficiência em códigos gerados por compiladores C. [11]

No caso do microcontrolador AT90CAN128 nota-se grande espaço na memória de programa (128kbytes), o que é importante, pois códigos gerados por tais compiladores são maiores que o equivalente gerado através da linguagem assembly.

Outra característica importante para uso da compilação em C é a presença de 32 registradores internos acessíveis diretamente pela Unidade Lógica Aritmética (ULA). Comparando-se com um microcontrolador 8051, que embora tenha vários registradores, somente um tem acesso direto pela ULA. Essa característica acelera o processamento de dados no microcontrolador.

Dentre os *softwares* para a programação do AVR, será utilizado o AVR Studio (figura 24) juntamente com o compilador C do pacote de aplicativos WinAVR (figura 25 e figura 26). Esses *softwares* estão disponíveis gratuitamente pela internet.

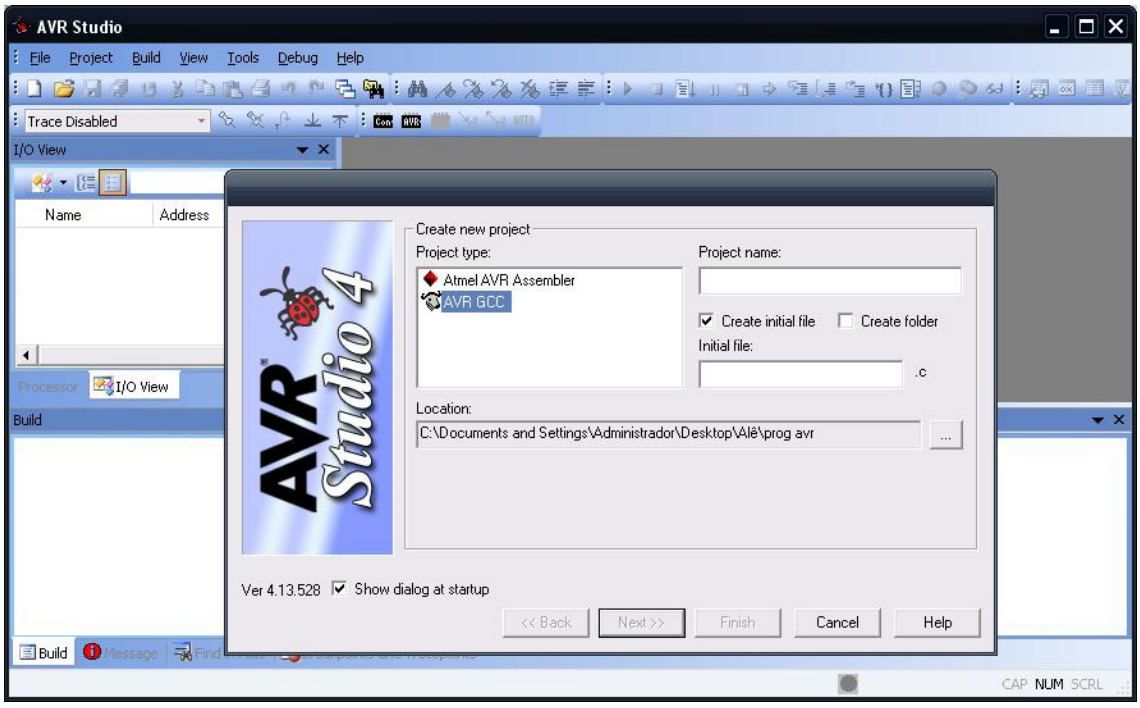

Figura 24 - *Software* de programação AVR Studio

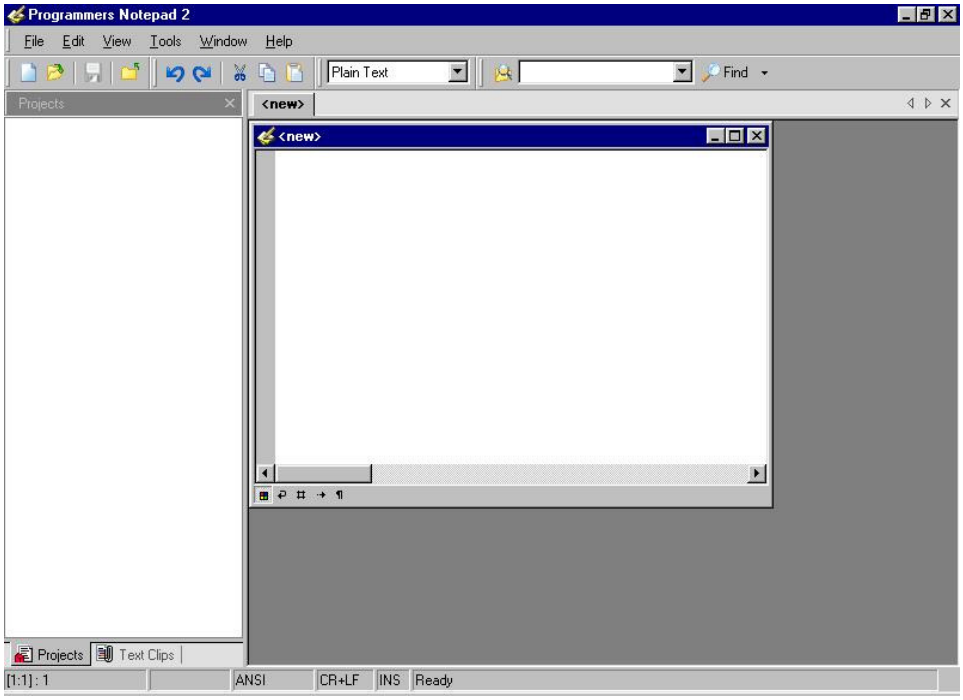

Figura 25 - Ambiente de programação do pacote de *softwares* WinAVR

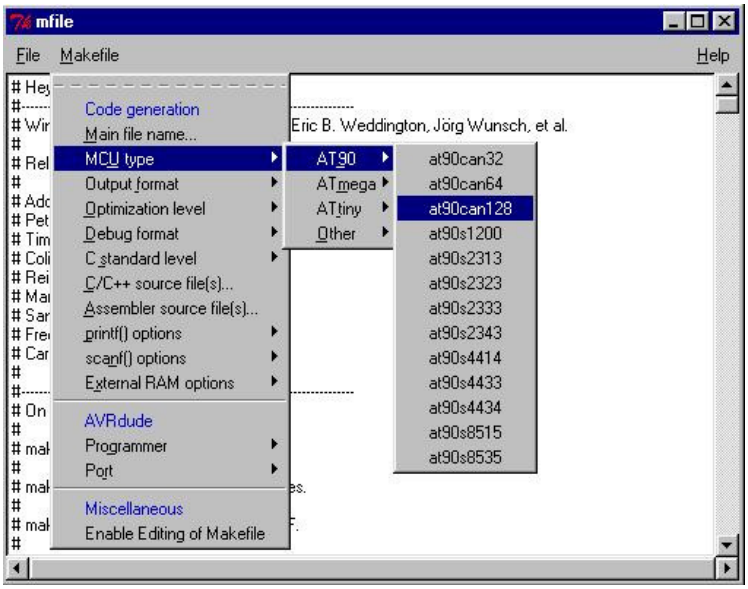

Figura 26 - Aplicativo para definir diretivas de compilação e programação de microcontroladores AVR

8. Construção da placa de controle CAN

Para fazer o uso da comunicação CAN, foi construído um circuito utilizando-se do mesmo tipo de microcontrolador utilizado na placa de desenvolvimento da Atmel e dispondo-se também de conectores para rede CAN, comunicação serial e conectores para acesso as portas do microcontrolador.

A figura 27 exibe o circuito da unidade de controle CAN, mostrando o microcontrolador AT90CAN128 (mesmos terminais do AT90CAN64) com fonte de clock gerado a partir do oscilador externo, estágio regulador de tensão com proteção contra inversão de polaridade. Neste circuito estão disponíveis também 6 conectores (J1 à J6) que dão acesso às portas do microcontrolador, um transceptor para porta serial MAX3232 e outro para porta de comunicação CAN ATA6660.

O conector J10 é utilizado para programação do microcontrolador.

Para correto casamento de impedância da rede CAN, e se esta placa estiver em um ponto extremo da rede, interligam-se os pinos do conector J7 para que um resistor de 120 seja ligado a rede.

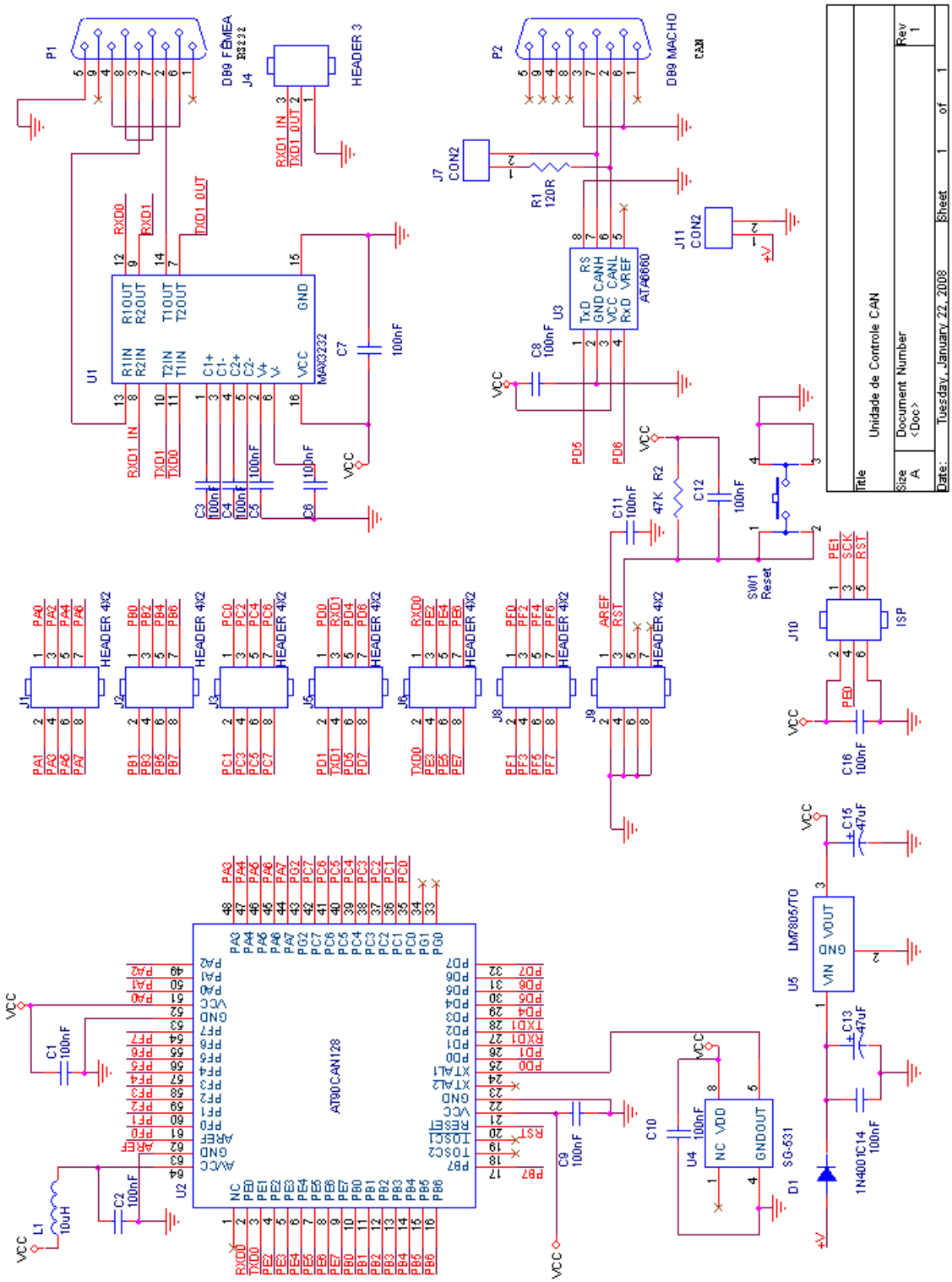

Figura 27 – Diagrama construído para segunda unidade de controle CAN

A partir do diagrama da unidade de controle CAN, foi desenvolvido circuito impresso, cujo *layout* é mostrado na figura 28 e construído pelo processo de fresagem utilizando uma placa de dupla face. Imagens da placa montada são apresentadas nas figuras 29 a 31.

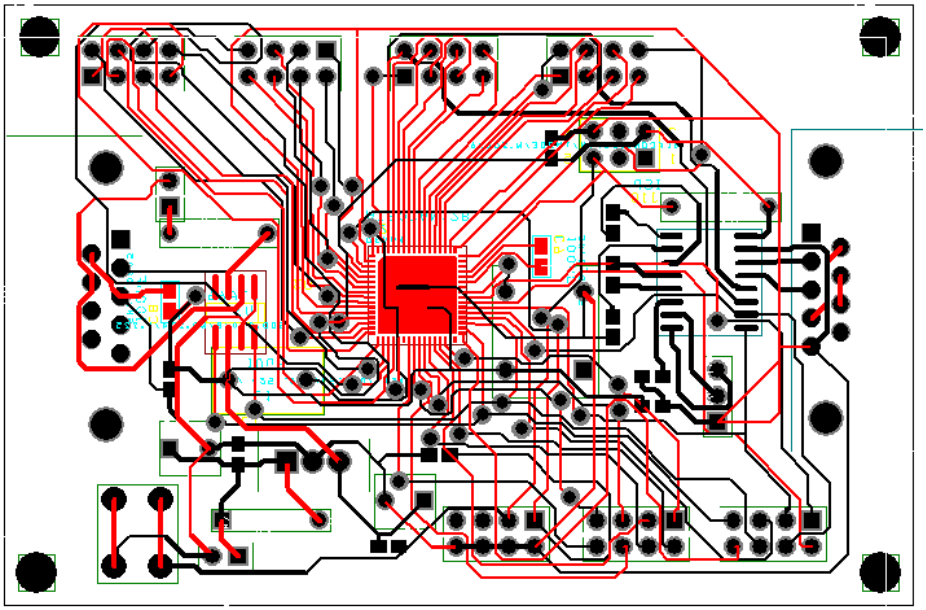

Figura 28 - Circuito impresso da unidade de controle CAN

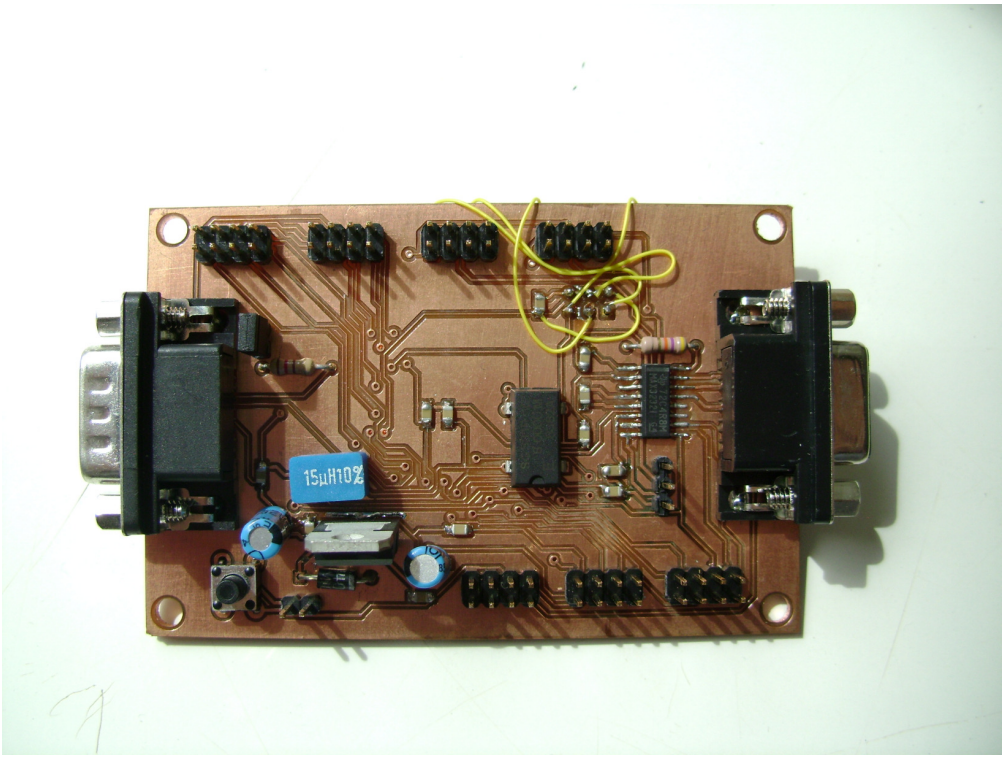

Figura 29 - Aspecto final da placa de controle CAN (face superior)

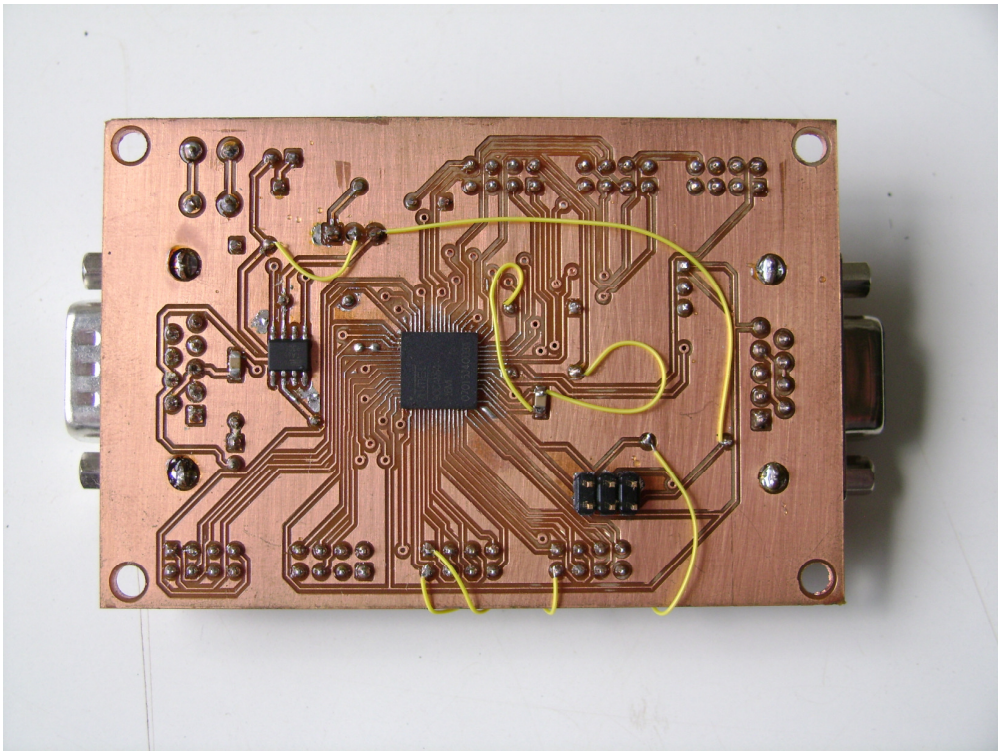

Figura 30 - Aspecto final da placa de controle CAN (face inferior)

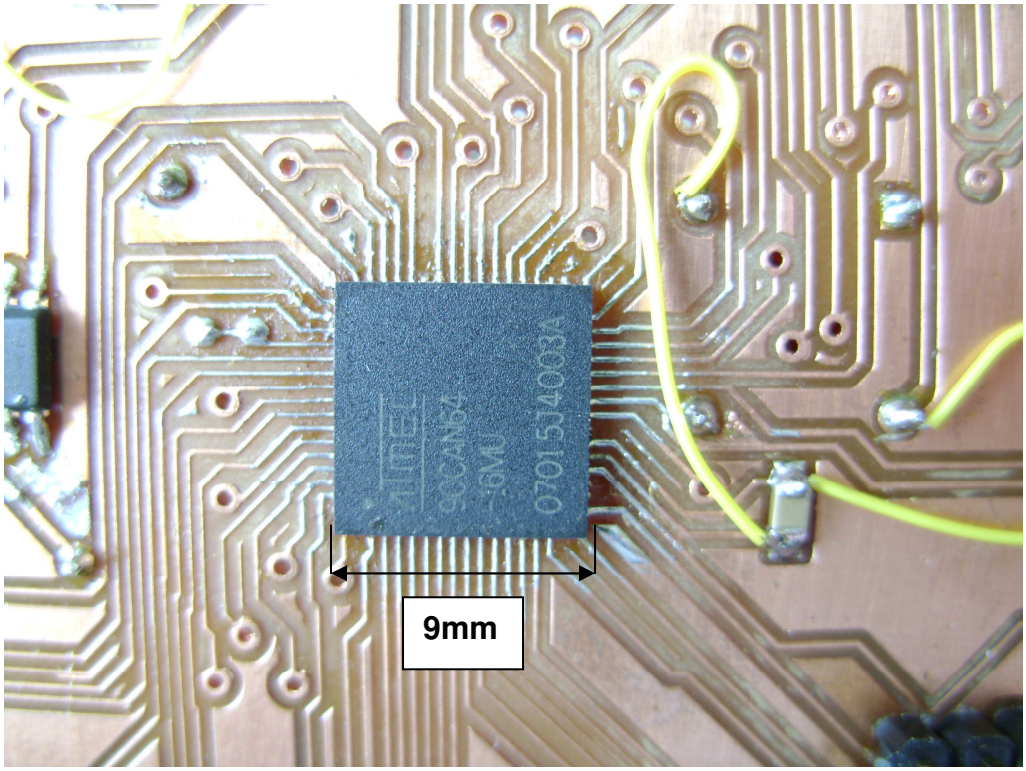

Figura 31 - Detalhe da soldagem do microcontrolador

Uma dificuldade encontrada na montagem da placa foi na soldagem do microcontrolador. Devido seu encapsulamento de dimensões reduzidas, tendo terminais muito concentrados (16 pinos em cada lado de 9mm), dificultaram a soldagem que foi feita

manualmente. Devido a erro no layout da placa descoberto após a montagem, o conector de programação ficou com a posição dos pinos invertida, o que obrigou a fazer sua soldagem na outra face da placa e com a utilização de fios (fios mostrados nas imagens da placa).

9. Configuração do sistema de teste

Para o teste da comunicação CAN, foi construído o sistema com configuração mostrada na figura 32. Esse sistema é constituído de duas unidades de controle, sendo a primeira (U.C. 1) o circuito construído indicado na Figura 29 e a segunda unidade (U.C. 2) o kit de desenvolvimento da Atmel (figura 20) . Esses módulos estão conectados via rede CAN. A U.C. 1 se comunica também através da porta serial com um microcomputador tipo PC e a unidade 2 é interligada aos sensores referencia de posição e as unidades de controle dos motores DC pelas pontes H.

A U.C.1 foi programada para operar como uma interface entre um microcomputador tipo PC ligado através de comunicação serial RS-232 e a U.C. 2, ligada através da rede CAN.

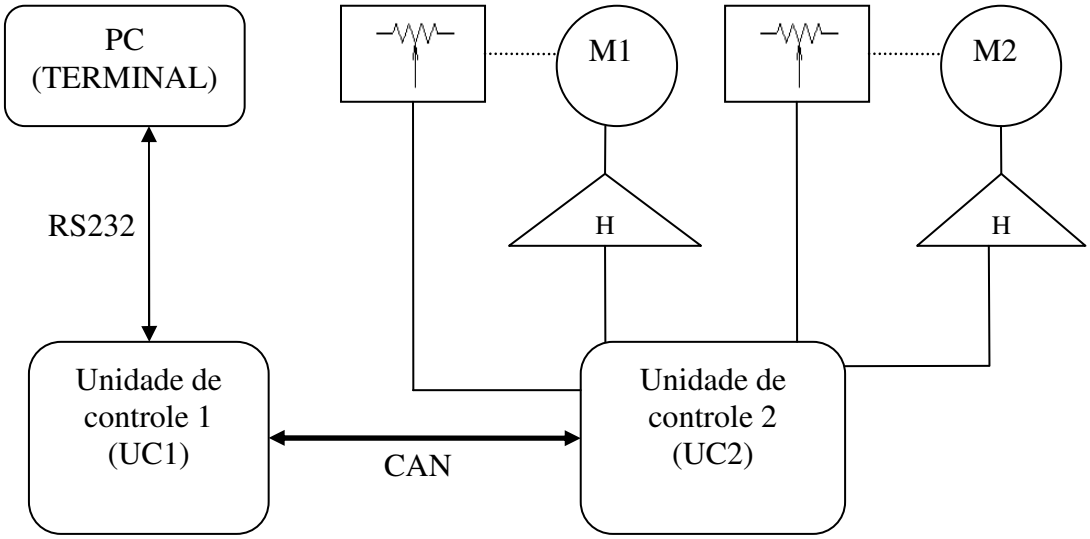

Figura 32 - Diagrama em blocos do sistema teste CAN

Através de um programa que emula um terminal de dados no microcomputador tipo PC, são enviados comandos à U.C. 1 que por sua vez os enviam pela rede CAN ao segundo módulo. Nesse segundo módulo estão conectados dois motores ligados através de pontes H comandadas com sinais PWM gerados pelo próprio microcontrolador. Para obter uma referencia de posicionamento desses motores, potenciômetros acoplados a esses operando como divisores de tensão têm seus sinais enviados ao conversor A/D do microcontrolador.

10. Programação dos microcontroladores

Para a programação foi utilizado a linguagem C com o compilador do pacote de programas *WinAVR*. O *software AVRDUDE* foi utilizado para gravação dos arquivos binários de programa nos microcontroladores e para simulações utilizou-se o programa *AVR Studio*.

A unidade de controle 1 foi programada para receber e enviar dados pela porta serial ligada a um computador e também enviar e receber dados pela rede CAN para se comunicar com o restante do sistema. Um fluxograma na figura 33 ilustra seu funcionamento.

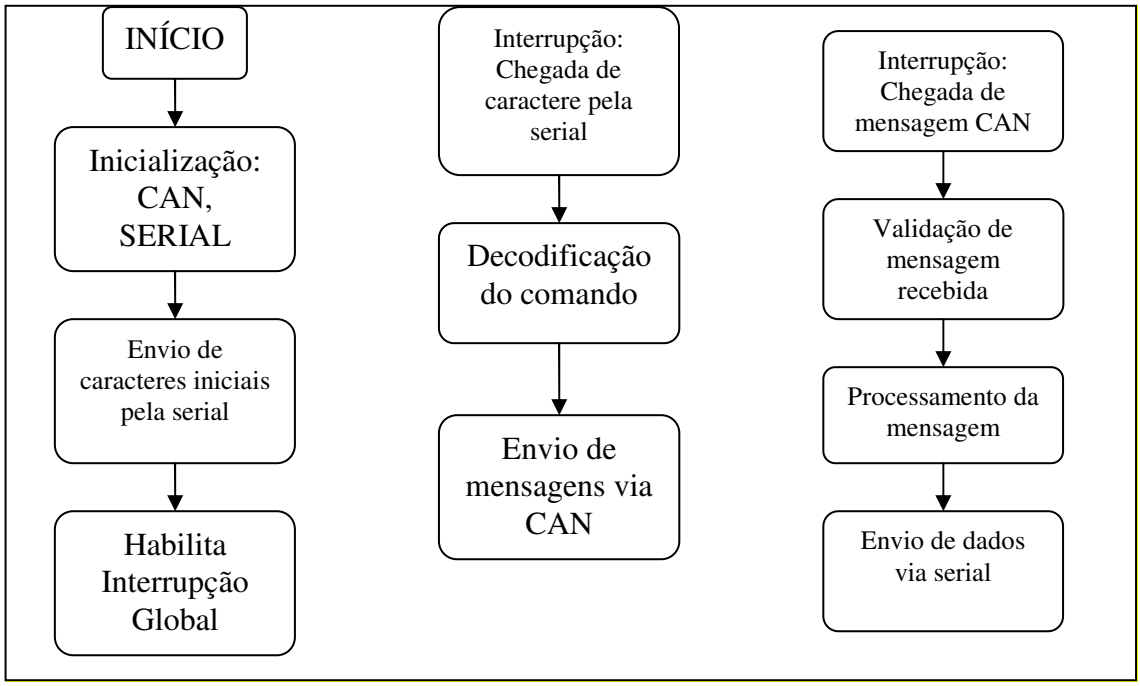

Figura 33 - Diagrama em blocos do funcionamento da unidade de controle 1

Já o programa da unidade de controle 2 opera de forma a atender os comandos vindos via CAN do módulo 1 e traduzi-los entre alterar os sinais PWM de controle dos motores ou efetuar a leitura da posição de cada motor através dos níveis de tensão dos potenciômetros. Um fluxograma na figura 34 ilustra seu funcionamento.

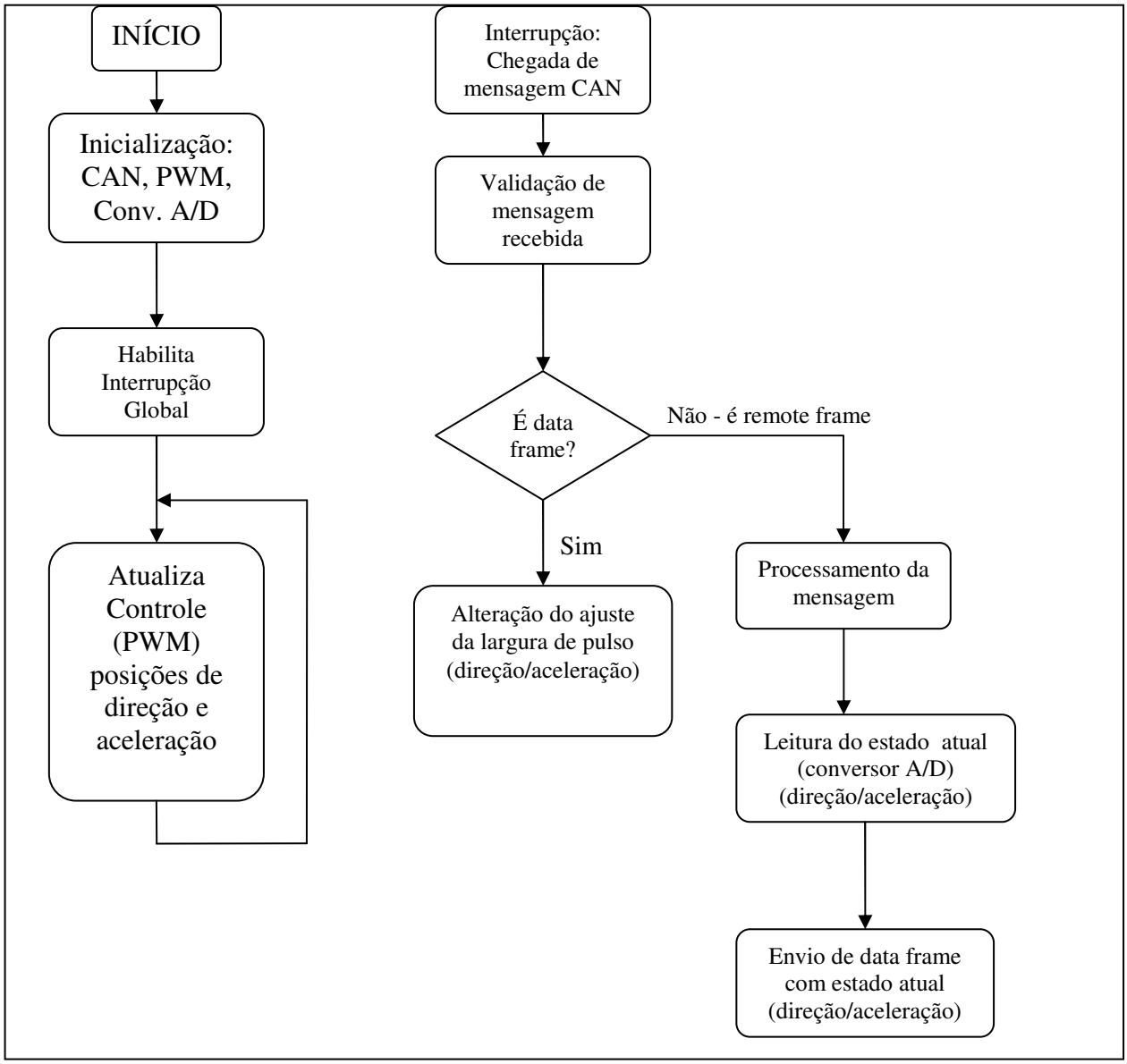

Figura 34 - Diagrama em blocos do funcionamento da unidade de controle 2

10.1 Operação do Controlador CAN no AT90CAN128/64

As informações de operação do microcontrolador foram obtidas em sua folha de dados. [10]

O controlador CAN do microcontrolador AT90CAN128/64 oferece suporte a mensagens com identificadores tanto de 11bits ou 29bits. Para os programas feitos foram usados somente com ID de 11bits visto as poucas mensagens necessárias para o funcionamento do sistema.

Um diagrama em blocos exibindo a estrutura do controlador CAN é mostrado na figura 35.

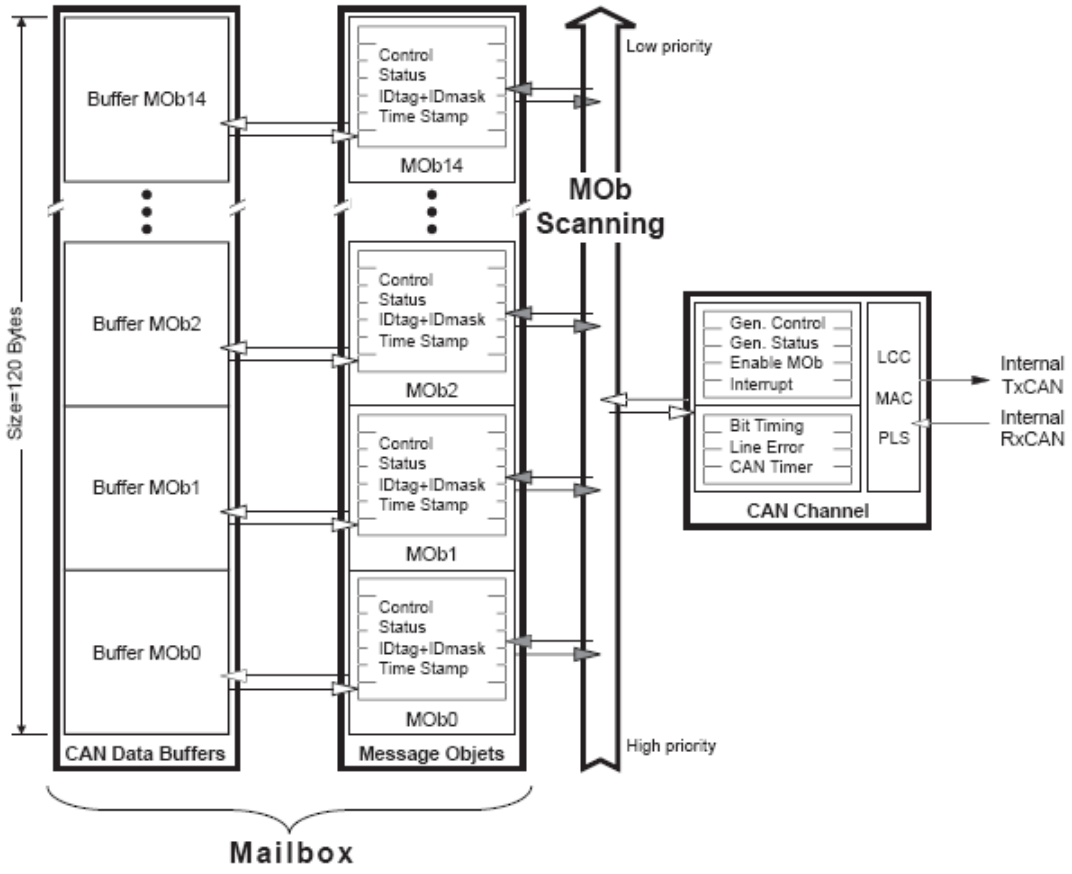

Figura 35 - Estrutura do controlador CAN no AT90CAN128/64

Nos controladores CAN dos micros AVR é utilizado o conceito de “caixa postal” para as quais as mensagens CAN que chegam ou irão ser transmitidas são armazenadas. Essa caixa postal é constituída de 15 objetos de mensagens (MOb). Para cada um desses objetos existem registradores exclusivos em que além do registrador do dado em si, existe também o registrador de controle, indicadores de estado, registradores de tempo e controle do filtro de mensagens.

O filtro de mensagens é um recurso utilizado para evitar que toda mensagem que chega ao microcontrolador seja processada, tornando o sistema mais ágil. Cada mensagem pode ter até 8 bytes de dados onde são armazenadas nos registradores (Data buffers).

10.2 Configuração e inicialização do controlador CAN

A seguir são descritos os registradores utilizados para configuração e inicialização do controlador CAN:

- CANGCON - CAN *General Control Register* :

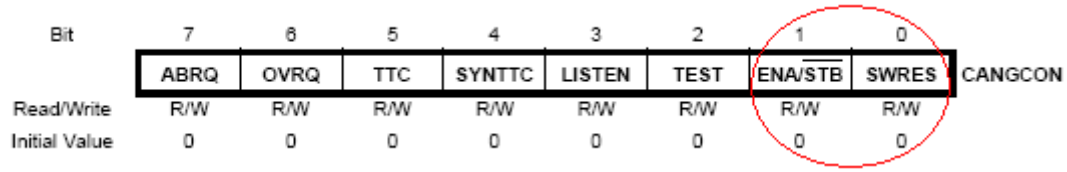

Figura 36 - Registrador CANGCON

Este é o registrador de controle geral do controlador CAN (figura 37). Nos programas realizados, foram utilizados os bits 0 e 1, tendo eles as seguintes funções:

SWRES - *Software Reset Request*

Escrevendo um valor 1 neste bit faz com que seja parado o controlador CAN.

ENA/STB: *Enable / Standby Mode*

Um valor 1 nesse bit ativa o controlador CAN, enquanto que um valor 0, deixa-o em modo de espera;

- CANPAGE - CAN Page MOb Register :

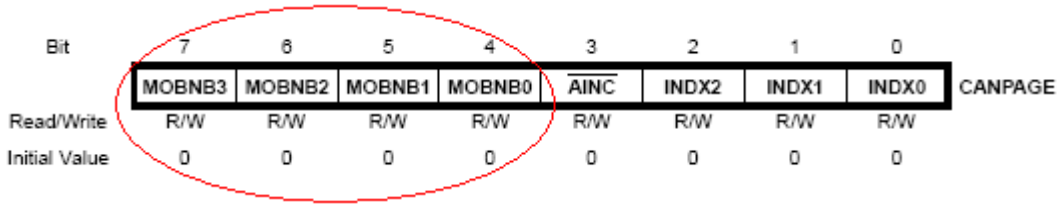

Figura 37 - Registrador CANPAGE

Os bits MOBNB0:3 do registrador CANPAGE (figura 37) são utilizados para seleção do MOb a ser utilizado e os valores possíveis são de 0 a 14.

Os bits INDX0:2 são usados para seleção de qual espaço dos 8 bytes serão gravados os dados, sendo 8 posições possíveis.

No programa não foi utilizado esse sistema de endereçamento. Com o bit AINC em 0 (o qual é seu valor inicial após um reset), o índice de dados é incrementado automaticamente quando uma informação é registrada.

- CANCDMOB - CAN MOb *Control and DLC Register*:

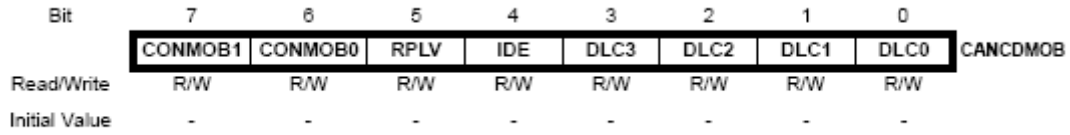

Figura 38 - Registrador CANCDMOB

Este registrador não é único, i.e, existe um registrador CANCDMOB para cada MOb disponível no microcontrolador (figura 38). Portanto para acessa-lo, deve-se antes saber de qual MOb está selecionado no momento através do CANPAGE. E isso vale também para os registradores CANSTMOB, CANIDT1 até CANIDT4, CANIDM1 até CANIDM4, CANSTML e CANSTMH.

Verifica-se que esses registradores não possuem um valor conhecido quando o microcontrolador é ligado, portando é necessário que todos esses registradores sejam inicializados com dados conhecidos.

Os bits DLC0:3 (*Data Length Code*) informa quantos bytes serão armazenados para os dados da mensagem CAN. Valores maiores que 8 (1000b) serão processados como 8.

O bit IDE (*Identifier Extension*), registra qual tipo de identificador de mensagem o MOb irá operar, sendo que se for valor 0 o identificador será formado por 11bits, ao passo que se for 1, o MOb irá operar com Identificação de 29bits. No primeiro caso serão possíveis 2^{11} tipos diferentes de mensagem e no segundo caso 2^{29} tipos diferentes de mensagens.

Para o programa utilizaram-se identificadores de 11bits.

Os Bits CONMOB1:0 (*Configuration of Message Object*) são usados para configurar os Mob's a respeito de recepção e transmissão de dados, de acordo com a tabela 2:

Tabela 2 - Configuração dos bits CONMOB1:0

CONMOB1	CONMOB0	Configuração do MOb
0	0	Desabilitado
0	1	Ativado para transmissão
1	0	Ativado para recepção
1	1	Ativado para recepção de frame buffer

No programa o MOb0 foi configurado para recepção de dados, enquanto que o MOb1 foi configurado para transmissão.

- CANSTMOB – (CAN MOb *Status Register*)

Bit	7	6	5	4	3	2	1	0	
	DLCW	TXOK	RXOK	BERR	SERR	CERR	FERR	AERR	CANSTMOB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	-	-	-	-	-	-	-	-	

Figura 39 - Registrador CANSTMOB

O registrador CANSTMOB indica o estado atual do Mob selecionado (figura 39):

a) DLCW - (*Data Length Code Warning*) Este bit avisa que uma mensagem recebida não tem o tamanho indicado no campo DLC;
b) TXOK - (*Transmit OK*) Este bit indica uma transmissão efetuada corretamente;
c) RXOK - (*Receive OK*) Este bit indica uma recepção correta de uma mensagem;
d) BERR - (*Bit Error*) Este bit indica que um bit monitorado não coincidiu com o bit enviado pelo microcontrolador;
e) SERR (*Stuff Error*) Este bit indica se houve um erro de bit stuff na comunicação CAN;
f) CERR (*CRC Error*) Este bit indica se houve um erro CRC na comunicação CAN;
g) FERR (*Form Error*) Este bit indica se houve um erro de forma na comunicação CAN e esse erro ocorre em violações de formas nos campos delimitadores do CRC e do ACK e no EOF da mensagem CAN;
h) AERR (*Acknowledgment Error*) Indicação de erro do bit ACK em uma comunicação;

- CANIDT1, CANIDT2, CANIDT3 e CANIDT4 (CAN *Identifier Tag Registers*):

Estes registradores possuem seus campos que dependem do modo dos identificadores CAN que for escolhido (11 ou 29bits), pelo bit IDE de CANCDMOB (figuras 40 e 41).

Para o caso de identificadores de 11 bits, os registradores possuem essa configuração:

V2.0 part A

Bit	15/7	14/6	13/5	12/4	11/3	10/2	9/1	8/0	
	-	-	-	-	-	RTRTAG	-	RB0TAG	CANIDT4
	-	-	-	-	-	-	-	-	CANIDT3
	IDT2	IDT1	IDT0	-	-	-	-	-	CANIDT2
	IDT10	IDT9	IDT8	IDT7	IDT6	IDT5	IDT4	IDT3	CANIDT1
Bit	31/23	30/22	29/21	28/20	27/19	26/18	25/17	24/16	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	-	-	-	-	-	-	-	-	

Figura 40 - Registradores CANIDTx para identificadores de 11bits

E para o caso de identificadores de 29 bits:

V2.0 part B

Bit	15/7	14/6	13/5	12/4	11/3	10/2	9/1	8/0	
	IDT4	IDT3	IDT2	IDT1	IDT0	RTRTAG	RB1TAG	RB0TAG	CANIDT4
	IDT12	IDT11	IDT10	IDT9	IDT8	IDT7	IDT6	IDT5	CANIDT3
	IDT20	IDT19	IDT18	IDT17	IDT16	IDT15	IDT14	IDT13	CANIDT2
	IDT28	IDT27	IDT26	IDT25	IDT24	IDT23	IDT22	IDT21	CANIDT1
Bit	31/23	30/22	29/21	28/20	27/19	26/18	25/17	24/16	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	-	-	-	-	-	-	-	-	

Figura 41 - Registradores CANIDTx para identificadores de 29bits

Sendo que foi utilizado apenas ID de 11 bits a configuração desses registradores tem a seguinte forma:

ID0 até ID10: onde é registrada a identificação da mensagem (data frame ou remote frame) a ser enviada.

RTRTAG (*Remote Transmission Request Tag*): Este bit é usado para distinguir um data frame (valor 0) de um remote frame (valor 1). Este bit é a atualizado quando é recebida uma nova mensagem, permitindo saber de qual tipo ela é.

- CANIDM1, CANIDM2, CANIDM3 e CANIDM4 (CAN *Identifier Mask Registers*):

  Para identificadores de 11bits, os registradores têm o seguinte aspecto:

V2.0 part A

Bit	15/7	14/6	13/5	12/4	11/3	10/2	9/1	8/0	
	-	-	-	-	-	RTRMSK	-	IDEMSK	CANIDM4
	-	-	-	-	-	-	-	-	CANIDM3
	IDMSK2	IDMSK1	IDMSK0	-	-	-	-	-	CANIDM2
	IDMSK10	IDMSK9	IDMSK8	IDMSK7	IDMSK6	IDMSK5	IDMSK4	IDMSK3	CANIDM1
Bit	31/23	30/22	29/21	28/20	27/19	26/18	25/17	24/16	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	-	-	-	-	-	-	-	-	

Figura 42 - Registradores CANIDMx para identificadores de 11bits

Esses registradores conjuntamente com CANIDT1:4 formam um filtro de mensagens CAN que irão ser processadas pelo microcontrolador (figura 42).

10.3 Filtragem de mensagens CAN

A seguir são mostrados alguns exemplos de como operam CANIDT e CANIDM na filtragem de mensagens. [10]:

a) Para aceitação de somente uma mensagem CAN (id único), os bits CANIDT e CANIDM terá as seguintes configurações (por ex. ID=0x317):

Tabela 3 – exemplo de filtragem para permissão de somente 1 identificador

Bit(x)	10	9	8	7	6	5	4	3	2	1	0
IDMSKx	1	1	1	1	1	1	1	1	1	1	1
IDTx (0x317)	0	1	1	0	0	0	1	0	1	1	1

Nota-se que todos bits IDMSKx têm valor 1.

b) Para aceitação de uma faixa de identificadores de mensagens CAN (por ex. ID de 0x310 até 0x317):

Tabela 4 - exemplo de filtragem para uma faixa de identificadores

Bit(x)	10	9	8	7	6	5	4	3	2	1	0
IDMSKx	1	1	1	1	1	1	1	1	0	0	0
IDTx (0x317)	0	1	1	0	0	0	1	0	X	X	X

X – não importa o valor

c) E para aceitação de qualquer mensagem CAN, basta fazer com que todos os bits IDMSKx sejam 0.

Para o programa foi filtro foi configurado para aceitar mensagens com identificação na faixa de 0x120 até 0x12F

- CANMSG (CAN *Data Message Register*)

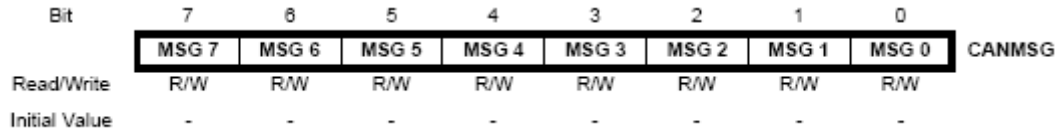

Figura 43 - Registrador CANMSG

Esse registrador armazena os dados da mensagem CAN (figura 43). Esse registrador opera em conjunto com os bits INDX0:2 para indicar qual byte esta sendo acessado e se for usado o índice automático, basta escrever 8 vezes nesse registrador para ter escrito nos 8 bytes disponíveis.

No programa feito, utilizou-se de somente 1 byte para os dados.

- CANBT1:3 (CAN *Baud Rate Setting*)

Através desses registradores se configura a taxa de transferência de dados pelo barramento CAN. O fabricante fornece através do *datasheet* uma tabela com valores padrão de velocidade e as respectivas configuração dos registradores (tabela 5). [10]

Tabela 5 - Configurações padrões para diferentes taxas de comunicação utilizando clock de 8MHz

$f_{clk_{Io}}$ (MHz)	CAN Baud Rate (Kbps)	Description			Segments				Registers		
		Sampling Point	TQ (µs)	Tbit (TQ)	Tprs (TQ)	Tph1 (TQ)	Tph2 (TQ)	Tsjw (TQ)	CANBT1	CANBT2	CANBT3
8.000	1000	75 %		x	- - - no data - - -						
			0.125	8	3	2	2	1	0x00	0x04	0x13
	500	75 %	0.125	16	7	4	4	1	0x00	0x0C	0x37
			0.250	8	3	2	2	1	0x02	0x04	0x13
	250	75 %	0.250	16	7	4	4	1	0x02	0x0C	0x37
			0.500	8	3	2	2	1	0x06	0x04	0x13
	200	75 %	0.250	20	8	6	5	1	0x02	0x0E	0x4B
			0.625	8	3	2	2	1	0x08	0x04	0x13
	125	75 %	0.500	16	7	4	4	1	0x06	0x0C	0x37
			1.000	8	3	2	2	1	0x0E	0x04	0x13
	100	75 %	0.625	16	7	4	4	1	0x08	0x0C	0x37
			1.250	8	3	2	2	1	0x12	0x04	0x13

Nos testes conseguiu-se através dos circuitos fazer comunicação à taxa de 1000kbps sem erros.

10.4 Mensagens CAN utilizadas

Na tabela 6 estão descritas as mensagens adotadas para o uso no sistema de teste CAN.

Tabela 6 - Mensagens CAN adotadas no programa

Identificador	*Data frame / Remote frame*	Função
0x120	*Remote frame*	Solicitar ao módulo 2 a posição atual da direção
0x121	*Remote frame*	Solicitar ao módulo 2 a posição atual da aceleração
0x120	*Data frame*	Envio pelo módulo 2 da posição atual da direção (1byte)
0x121	*Data frame*	Envio pelo módulo 2 da posição atual da aceleração (1byte)
0x123	*Data frame*	Envio pelo módulo 1 da posição desejada de direção (1byte)
0x124	*Data frame*	Envio pelo módulo 1 da posição desejada de aceleração (1byte)

10.5 Comunicação serial no AT90CAN128/64

Para o teste do programa, optou-se por gerar os dados de controle através de um microcomputador tipo PC, emulando um terminal de dados, conectado através da porta serial à porta serial do microcontrolador CAN (especificar se é sua placa ou a placa de desenvolvimento).

A seguinte configuração foi utilizada no terminal de dados:

- Palavras de 8 bits
- 1 bit de parada
- Sem paridade
- Comunicação assíncrona
- Taxa de 9600bps

O modo de interrupção para porta serial foi ativado, para que o módulo 1 esteja sempre no aguardo de comandos vindos via programa terminal do computador.

10.6 Conversor A/D

O conversor A/D do AT90CAN128/64 utiliza da técnica da aproximação sucessiva para realização da conversão. Esse é um modo rápido de conversão, porém, são mais afetados por ruídos que os modos integrativos de conversão.

São possíveis conversões de até 8 sinais diferentes pelo microcontrolador. Isso é feito, através de chaves analógicas e através de um processo de multiplexação, converte um canal por vez, comparando-o com uma tensão interna de referência.

Para obter o posicionamento de cada motor de controle, foram utilizados potenciômetros acoplados a esses, que enviam um nível de tensão referente à posição atual, do comando da direção e da aceleração do barco.

Para poder processar esses níveis de tensão, utilizou-se de dois canais do conversor A/D do microcontrolador.

Embora o conversor seja de 10bits de resolução, foi optado por utilizar apenas os 8 bits mais significativos da conversão para o processamento, tendo assim 256 níveis binários distintos referentes a posições possíveis do potenciômetro o que é suficiente para a aplicação, e ainda faz com que se reduza o efeito de ruídos na conversão, eliminando bits menos significativos.

10.7 Gerador de sinais PWM

O microcontrolador AT90CAN128/64 possui 3 temporizadores que podem ser configurados como geradores de sinais PWM, sendo 2 de 8 bits e um de 16 bits. Para o programa, utilizou-se os dois temporizadores de 8 bits.

Na figura 44 é exibido um diagrama em blocos de um dos temporizadores (Timer0) e sua operação é semelhante ao segundo temporizador de 8 bits.

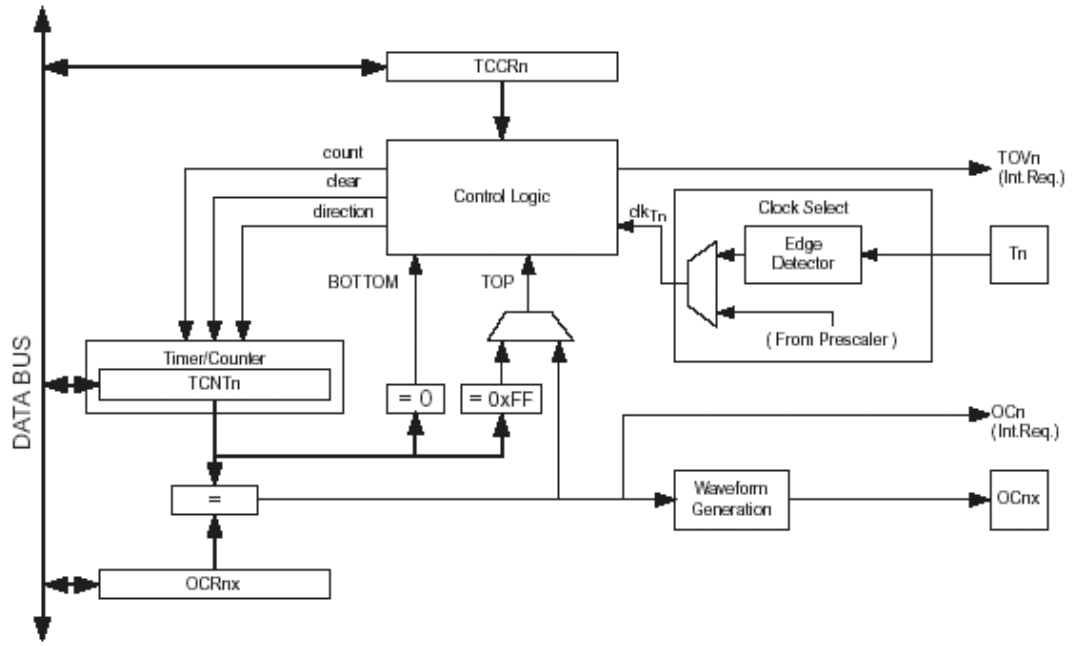

Figura 44 - Diagrama em blocos do temporizador TIMER-0 do microcontrolador AVR AT90CAN64/128

Sua configuração é feita a partir dos bits dos Registradores TCCRn, configurados de acordo com as informações do *datasheet* para operação "*Fast PWM*" .

O registrador TCNTn armazena a contagem de tempo e é comparado através do registrador OCRnx. O resultado desta comparação é enviado ao bloco gerador de forma de onda que por sua vez envia o sinal ao pino de saída OCnx.

Para o controle dos dois motores, os módulos PWM foram configurados para as seguintes formas de sinais:

a) Período do pulso em torno de 20ms
b) Ciclo de trabalho do pulso PWM: para o comando PWM utilizado, pulsos de 1ms de largura fará com que o motor gire para uma direção e pulso de 2ms de largura fará com que o motor gire para o lado oposto. Pulsos de 1,5ms faz com que o motor fique parado.

Esses sinais são posteriormente convertidos em um circuito a sinais PWM de alta freqüência e controle de reversão dos motores através do circuito ponte H.

11. Testes do sistema

Para os testes do sistema, os circuitos foram montados em um triciclo adaptado que esta sendo desenvolvido para simular os controles de direção e aceleração do barco. Esse veículo é usado para simulação da plataforma, onde os testes são feitos em terra, economizando em deslocamento até uma represa para testes no barco. O triciclo é controlado por dois

motores CC, sendo um usado para tração das rodas traseiras e outro para controle de direção das rodas (figura 45).

Figura 45 - Triciclo desenvolvido para testes do sistema de comando da embarcação.

Ligado ao eixo de direção, foi instalado um sensor de posição que possui como saída um nível CC referente ao ângulo de direção da roda frontal (figura 46).

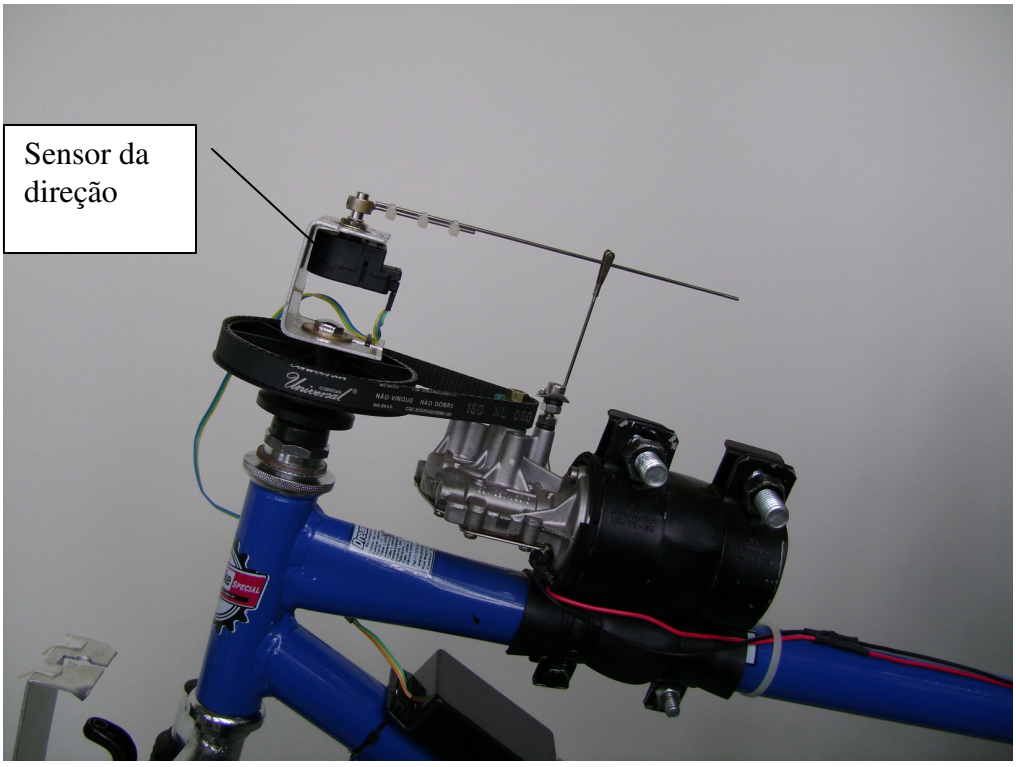

Figura 46 - Detalhe do sensor de posição do controle de direção

Na figura 47 é exibido as duas unidades de controle em fase de teste

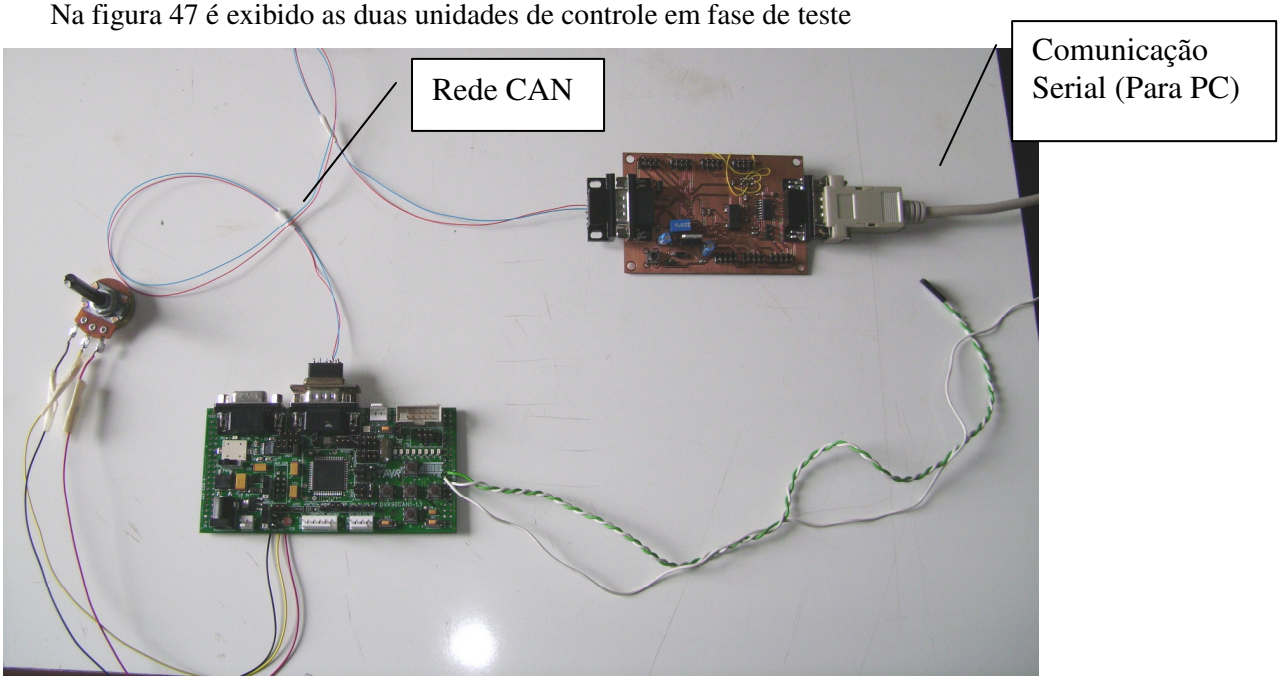

Figura 47 - Teste das unidades de controle

Na figura 48 são mostrados a unidades de controle, com ligada aos motores e sensores de posição e outra sendo a interface entre porta serial e rede CAN:

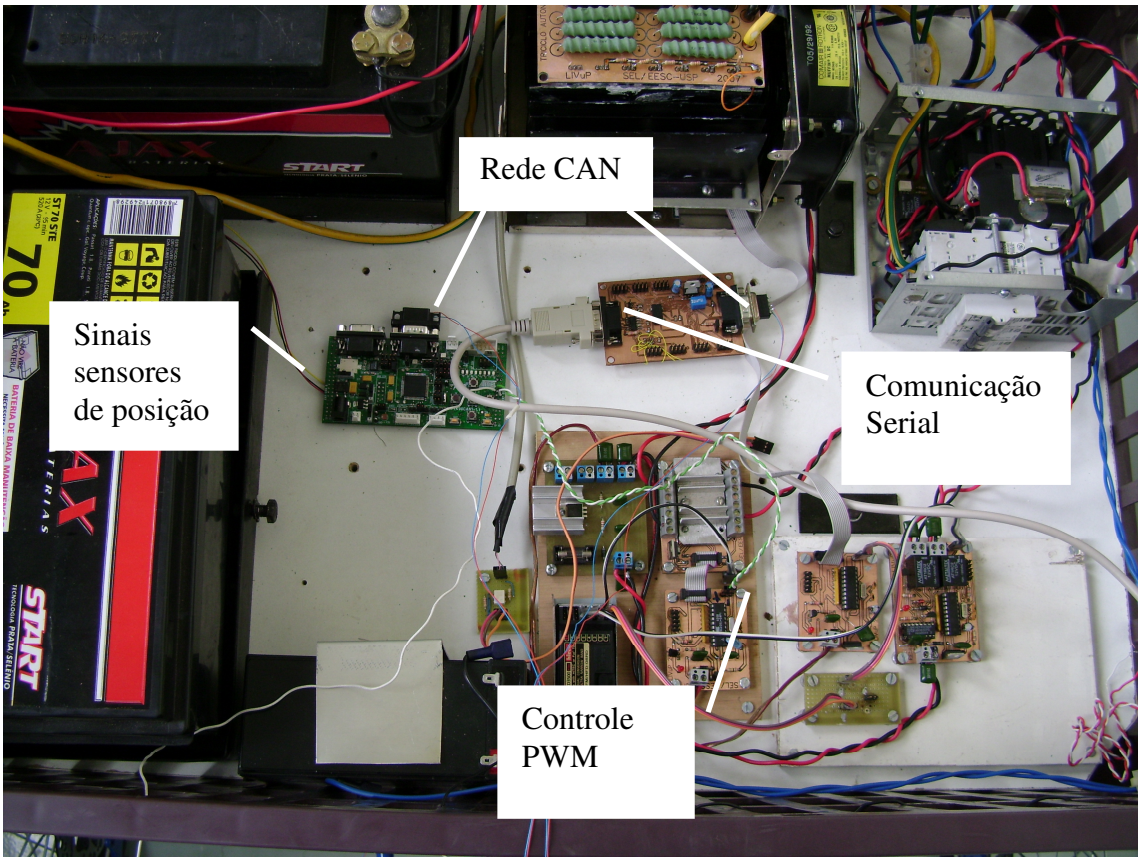

Figura 48 - Teste das unidades de controle no triciclo

Ao ligar o sistema, a U.C. 1 tem suas portas seriais e CAN configuradas e é enviada ao programa terminal do PC uma mensagem indicando as opções possíveis de operação (figura 49):

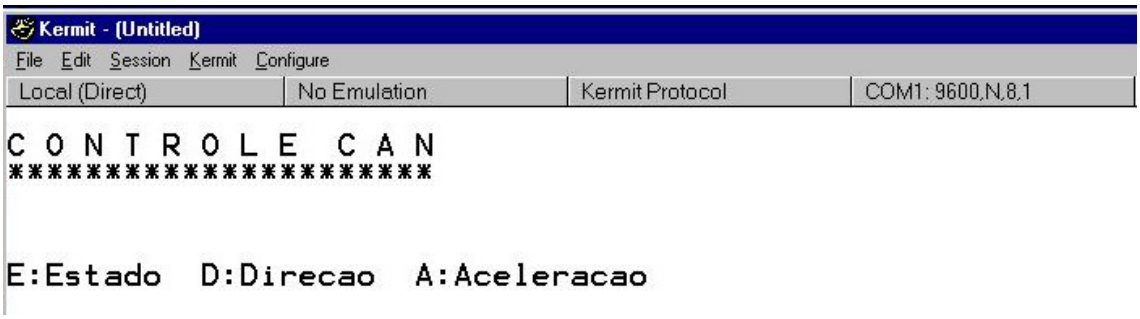

Figura 49 - Tela inicial do programa

Ao mesmo tempo, é inicializado na U.C. 2 o controlador CAN , as saídas PWM e o conversor A/D. Então a direção é posicionada no ponto central e a aceleração modo parado (central).

Ao pressionar a tecla E no teclado, é enviado um comando ao módulo 1 que por sua vez envia a requisição ao módulo 2 via CAN para a leitura dos estados atuais dos controles de direção e aceleração. Esta informação é então retornada ao módulo 1 e então é enviada ao PC e exibida na tela (figura 50).

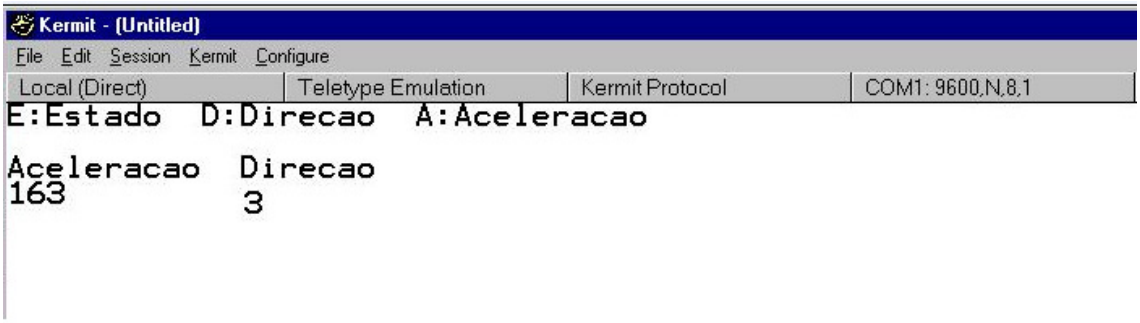

Figura 50 - Visualização do Estado atual dos controles

Se for pressionada a tecla D, a informação da direção atual é mostrada e pode ser modificada pelo usuário (figura 51).

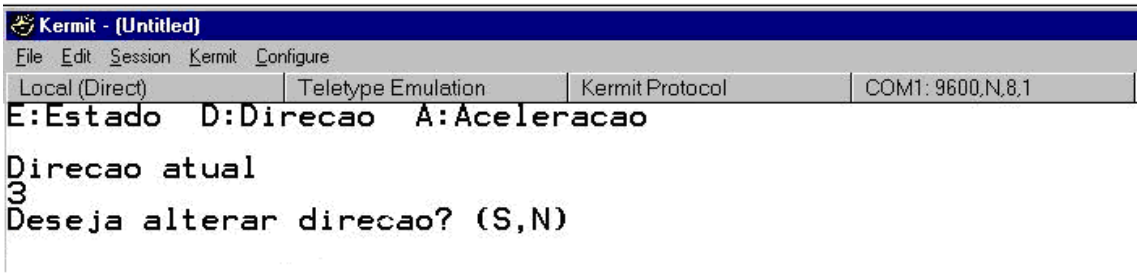

Figura 51 - Ajuste do controle de direção

De forma semelhante para no caso de verificar e/ou modificar o valor da aceleração (figura 52).

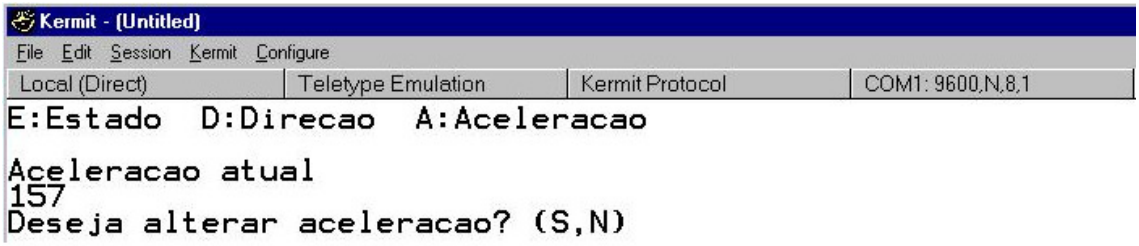

Figura 52 - Ajuste do controle de aceleração

Ao aceitar a modificação em ambas as opções, o programa do módulo 1 aguarda 3 números serem digitados para requisição de nova direção ou aceleração. E o programa só aceita valores de zero(000) até 255, sendo que valores fora dessa faixa serão ignorados.

Pelos testes, foi possível a realização da leitura dos estados, bem como a alteração dos mesmos.

12. Conclusões

Com o trabalho desenvolvido, pode-se desenvolver um sistema de controle para direção e aceleração do veículo utilizando-se a rede CAN.

Com a rede CAN, pode-se projetar uma arquitetura distribuída do sistema da plataforma, tornando-a flexível para futuras modificações e expansões.

Foi verificado um funcionamento esperado para rede CAN, onde dados podem ser tanto transmitidos como recebidos pelas unidades de controle além do monitoramento do sistema via aplicativo terminal no microcomputador tipo PC ligado a porta serial.

Os testes iniciais foram realizados com comandos do usuário enviados via porta serial do microcomputador, sem malha de controle, porém futuramente esse comando passará a ser de modo autônomo, utilizando-se da lógica fuzzy.

Novos instrumentos poderão ser adicionados à rede CAN, como GPS e bússola eletrônica, instrumentos que serão necessários para o comando autônomo do veículo.

13. Anexos

13.1 Programa da U.C. 1

Referências: [12] [13]

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>


unsigned char aceleracao=0, direcao=0;dir_rx_ok=0; acel_rx_ok=0;



//inicialização da porta CAN
//MOb 0 para receber dados
//Mob 1 para enviar dados


void can_ini(void)        {
                    unsigned char num_mob, num_dados;
                    CANGCON = 0X01; //RESETA CAN

                    //zerar todas as caixas de mensagens

                    for(num_mob=0;num_mob<15;num_mob++)
                    {
                    CANPAGE= num_mob<<4;  //seleciona o Mob
                    CANCDMOB= 0x00; //zera os bits de controle
                    CANSTMOB= 0X00; //zera os bits de Indicação de estado
                    //zerar os tags de Identificação:
                    CANIDT1=0;
                    CANIDT2=0;
                    CANIDT3=0;
                    CANIDT4=0;

                    //zerar as mascaras de Identificação:
                    CANIDM1=0;
                    CANIDM2=0;
                    CANIDM3=0;
                    CANIDM4=0;

                    for(num_dados=0; num_dados<8; num_dados ++)
                    CANMSG=0; //limpa os registradores de mensagem de todos Mob's
                    }

                    //configurar velocidade da rede CAN para 500 kbps com cristal de 8Mhz:

                    CANBT1=0x00;
                    CANBT2=0X0c;
```

```
        CANBT3=0X37;

        CANGCON=0X02; //inicia controlador CAN

        //Configuração do mob 0 para recepção de mensagens

        CANPAGE = (0<<4);
        CANSTMOB=0;
        CANCDMOB=0;

        //filtragem de mensagens entre as id's 0x120 e 0x12f:

        CANIDT1=0X24;
        CANIDT2=0X00;

        CANIDM1 =0XFE;
        CANIDM2 &= ~0XE0;
        CANIDM4 = 0;

        //      Configuração do mob0

        CANIDT4 &= ~0X04;  //zera bit rtr
        CANCDMOB |= 1;              //Recepção de 1 bytes
        CANCDMOB |=0X80;            //Recepção sem buffer

        //configuração do Mob 1

        CANPAGE=(1<<4);
        CANSTMOB=0X00;
        CANCDMOB=0X00;

        CANIDT1=0X24;
        CANIDT2=0X00;

        CANIDM1 =0XFE;
        CANIDM2 &= ~0XE0;
        CANIDM4 = 0;

        //configura Interrupção de CAN

        CANPAGE = (0<<4);

        CANIE2 |=0X01;
        CANGIE = ((1<<ENRX) | (1<<ENIT));


    }


//função para enviar um remote frame pela CAN
void RF_CAN(unsigned int id){
        CANPAGE=(1<<4);
        CANIDT2=(char)(id<<5);
        CANIDT1=(char)(id>>3);
        CANIDT4|=(1<<RTRTAG); //seta bit RTR
```

```
        CANSTMOB=0;
        CANCDMOB=0;//ZERO BYTES DE DADOS ENVIADOS
        CANCDMOB |=0X40; //ENVIA MSG
        CANPAGE=(0<<4); //retorna ao modo de recepção
        }


//função para enviar um data frame pela can
void DF_CAN(unsigned int id, unsigned int dado){
        CANPAGE=(1<<4);
        CANIDT2=(char)(id<<5);
        CANIDT1=(char)(id>>3);
        CANIDT4&=(0<<RTRTAG); //zera bit RTR
        CANMSG=(char)(dado);
        CANSTMOB=0;
        CANCDMOB=1;//transmissão de 1 byte de dados
        CANCDMOB |=0X40; //ENVIA MSG
        CANPAGE=(0<<4);//retorna ao modo de recepção
        }


//função que retorna um valor inteiro recebido pela serial
int USART_Rx_num(void) {
char s[3]; //3 caracteres + '\0'
int i,v;
for(i=0;i<3;i++){
        //Aguarda chegada de dado
        while ( !(UCSR0A & (1<<RXC0)) );
        s[i]=UDR0;
        }
        //Recebe o dado do buffer
v=atoi(s);
if((v>255)|(v<0))
return -1;
return v;
}

//função que retorna um caractere recebido pela serial
char USART_Rx(void) {
//Aguarda chegada de dado
while ( !(UCSR0A & (1<<RXC0)) );
//Recebe o dado do buffer
return UDR0;
}


//transmissão de 1 caractere via serial
void USART_Tx(char c) {
//Verificar se o buffer da serial está vazio
while ( !(UCSR0A & (1<<UDRE0)) );
//coloca o dado no buffer e envia
UDR0 = c;
}
```

```
//transmissão de string pela serial usando a função USART_Tx()

void str_tx(char *saida)
        {
        int i;
        i=0;
                do{
                USART_Tx (saida[i]);
                i++;
                }while(saida[i]!='\0');

        }

void menu0(void){
        str_tx("\n\n\n\rE:Estado  D:Direcao  A:Aceleracao\n\r");
        }


void acel(void)  {
        int valor;
        char s[3],aux;
        RF_CAN(0x121);
        while((acel_rx_ok==0)||((CANGSTA&0X10)==0X10));
        cli();
        str_tx("\n\rAceleracao atual\n\r");
        itoa(aceleracao,s,10);
        str_tx(s);
        str_tx("\n\rDeseja alterar aceleracao? (S,N)\n\r");
        aux=USART_Rx();
        if((aux=='S')||(aux=='s'))
                {str_tx("entre com o novo valor");
                valor=USART_Rx_num();
                if(valor>-1)
                DF_CAN(0x124,valor);
                }
        else if((aux=='N')||(aux=='n'))
                {str_tx("cancelado");

                }
        else
        str_tx("comando invalido");
        }

void le_pos(void){
        //int a,d;
        char s[3];
        RF_CAN(0x120);
        while((dir_rx_ok==0)||((CANGSTA&0X10)==0X10));
        RF_CAN(0x121);
        while((acel_rx_ok==0)||((CANGSTA&0X10)==0X10));
        str_tx("\n\rAceleracao  Direcao\n\r");
        itoa(aceleracao,s,10);
        str_tx(s);
        str_tx("          ");
        itoa(direcao,s,10);
```

```
        str_tx(s);
        sei();
        }

void direc(void)        {
        int valor;
        char s[3],aux;
        RF_CAN(0x120);
        while((dir_rx_ok==0)||((CANGSTA&0X10)==0X10));
        cli();
        str_tx("\n\rDirecao atual\n\r");
        itoa(direcao,s,10);
        str_tx(s);
        str_tx("\n\rDeseja alterar direcao? (S,N)\n\r");
        aux=USART_Rx();
        if((aux=='S')||(aux=='s'))
                {str_tx("entre com o novo valor");
                valor=USART_Rx_num();
                if(valor>-1)
                DF_CAN(0x123,valor);
                }
        else if((aux=='N')||(aux=='n'))
                {str_tx("cancelado");

                }
        else
        str_tx("comando invalido");

        }


ISR(USART0_RX_vect)

        {
        cli();
        char comando;
        comando=UDR0; //lê o caracter que chegou à serial

switch(comando){
        case('A'):      {sei(); acel();}  break;
        case('D'):      {sei(); direc();} break;
        case('E'):      {sei();le_pos();}       break;
        }

        menu0();
        sei();
        }


//rotina da interrupção CAN

ISR(CANIT_vect){
//void canint(void){
                cli();
                unsigned int id;
```

```
        //alt_led();
        CANPAGE= (0<<4); //seleciona mob0
        if((CANSTMOB & 0X20) == 0X20) //verificaçÃo de msg ok
        {
        id=(((int)(CANIDT2)) >> 5) + (((int)(CANIDT1))<<3);


        switch(id)
                {
                case(0x120):    if((CANIDT4 & 0x04) == 0x00)
                                        {direcao=(int)(CANMSG);
                                        dir_rx_ok=1;
                                        } break;
                case(0x121):    if((CANIDT4 & 0x04) == 0x00)
                                        {aceleracao=(int)(CANMSG);
                                        acel_rx_ok=1;
                                        }break;
                }

        }
        CANPAGE=(0<<4);
        CANSTMOB=0;
        CANEN2 |=(1<<0);
        CANCDMOB=1; //recebe 1 bytes no max
        CANCDMOB |=0X80; //ativa recepção
        CANPAGE=(1<<4);
        sei();

}


int main (void){

        //inicialização da CAN
        can_ini();


        //inicialização da porta serial
        //PALAVRAS DE 8 BITS, 1 STOP BIT, S/ PARIDADE, comunicação assíncrona
        UCSR0C =((0<<UMSEL0)|(1<<UCSZ01)|(1<<UCSZ00));


        //Setar baud rate 9600 (sem o double speed) com clock em 8MHz
        UBRR0H=0;
        UBRR0L=51;

        //ATIVAR TRANSMISSÃO, recepção e interrupção via serial
        UCSR0B =((1<<TXEN0)|(1<<RXEN0)|(1<<RXCIE0));

        str_tx("\n\rC O N T R O L E  C A N\n\r");
        str_tx("**********************\n\r");

        menu0();
        sei();
        return 0;
```

```
}
```

13.2 Programa da U.C. 2

Referências: [12] [13]

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>


unsigned char direcao=0,dir_rq=128,aceleracao=0,acel_rq=128;


//inicialização do conversor AD
void ad_ini(void) {
ADCSRA = _BV(ADEN) | _BV(ADPS2) | _BV(ADPS1);
ADMUX = _BV(REFS0);
}


//inicialização da CAN
void can_ini(void)        {
                    unsigned char num_mob, num_dados;
                    CANGCON = 0X01; //para controlador CAN

                    //zerar todos MOb's

                    for(num_mob=0;num_mob<15;num_mob++)
                    {
                    CANPAGE= num_mob<<4;  //seleciona o Mob
                    CANCDMOB= 0x00; //zerar os bits de controle
                    CANSTMOB= 0X00; //zerar os bits de indicaçÃo de estado
                    //zerar os tags de identificaçÃo:
                    CANIDT1=0;
                    CANIDT2=0;
                    CANIDT3=0;
                    CANIDT4=0;

                    //zerar as mascaras de identificaçÃo:
                    CANIDM1=0;
                    CANIDM2=0;
                    CANIDM3=0;
                    CANIDM4=0;

                    for(num_dados=0; num_dados<8; num_dados ++)
                    CANMSG=0; //limpa os registradores de mensagem de todos Mob's
                    }

                    //configurar velocidade da rede CAN para 500 kbps com cristal DE 8Mhz:

                    CANBT1=0X00;
                    CANBT2=0X0c;
                    CANBT3=0X37;
```

```
        CANGCON=0X02; //inicia controlador CAN

        //configuraçÃo do mob 0 para recepçao de mensagens

        CANPAGE = (0<<4);
        CANSTMOB=0;
        CANCDMOB=0;

        //filtragem de mensagens entre as id's 0x120 e 0x12f:

        CANIDT1=0X24;
        CANIDT2=0X00;

        CANIDM1 =0XFE;
        CANIDM2 &= ~0XE0;
        CANIDM4 = 0;

        //      configuraçÃo do mob0

        CANIDT4 &= ~0X04;  //zera bit rtr
        CANCDMOB |= 1;      //recepção de 1 byte
        CANCDMOB |=0X80; //recepção sem buffer

        //configuração do Mob 1

        CANPAGE=(1<<4);
        CANSTMOB=0X00;
        CANCDMOB=0X00;

        CANIDT1=0X24;
        CANIDT2=0X00;

        CANIDM1 =0XFE;
        CANIDM2 &= ~0XE0;
        CANIDM4 = 0;
        CANPAGE = (0<<4);

        //configura interrupçÃo de CAN
        CANIE2 |=0X01;
        CANGIE = ((1<<ENRX) | (1<<ENIT));


    }


//função para enviar um data frame pela can
void DF_CAN(unsigned int id, unsigned int dado){
    CANPAGE=(1<<4);
    CANIDT2=(char)(id<<5);
    CANIDT1=(char)(id>>3);
    CANIDT4&=(0<<RTRTAG); //zera bit RTR
    CANMSG=(char)(dado);
    CANSTMOB=0;
```

```
        CANCDMOB=1;//transmissão de 1 byte de dados
        CANCDMOB |=0X40; //ENVIA MSG
        CANPAGE=(0<<4);
        }


ISR(CANIT_vect){
                cli();
                unsigned int id;
                //alt_led();
                CANPAGE= (0<<4); //seleciona mob0
                if((CANSTMOB & 0X20) == 0X20) //verificação de msg ok
                {
                id=(((int)(CANIDT2)) >> 5) + (((int)(CANIDT1))<<3);


                switch(id)
                        {
                        case(0x120):    if ( (CANIDT4 & 0x04) == 0x04) //verifica que é um
remote frame
                                        {
                        DF_CAN(0x120,direcao); //envia de volta valor da direção atual
                                        }
                                        break;
                        case(0x121):    if ( (CANIDT4 & 0x04) == 0x04)
                                        {
                DF_CAN(0x121,aceleracao); //envia de volta valor aceleração atual
                                        }
                                        break;
                        case(0x123):    if ( (CANIDT4 & 0x04) == 0x00) //verificação de data
frame
                                        {dir_rq=CANMSG;}break;
                        case(0x124):    if ( (CANIDT4 & 0x04) == 0x00)
                                        {acel_rq=CANMSG;}break;
                        }

                }

                CANPAGE=(0<<4);
                CANSTMOB=0;
                CANEN2 |=(1<<0);
                CANCDMOB=1; //recebe 1 bytes
                CANCDMOB |=0X80; //ativa recepção
                sei();

        }


//função para ler um canal AD
void ad_conv(unsigned char canal, unsigned char *dado){
ADMUX = (1<<REFS0) |(canal & 0x0F)|(1<<ADLAR);
ADCSRA |= (0x01<<ADSC);
while((ADCSRA & (0x40))); //aguarda fim da conversão
*dado = ADCH;//registra bits mais significativos(8BITS)
}
```

```
void pos_ac(unsigned char pos){
if(pos<(direcao-10))
                {OCR0A=8;
                TCCR0A|=(1<<CS22)|(1<<CS21)|(1<<CS20);
                }
        else if(pos>(direcao+10))
                {OCR0A=14;
                TCCR0A|=(1<<CS22)|(1<<CS21)|(1<<CS20);
                }
        else if((pos>=(direcao-10))&&(pos<=(direcao+10)))
                {OCR0A=11;
                TCCR0A|=(1<<CS22)|(1<<CS21)|(1<<CS20);
                }
}
void pos_dir(unsigned char pos){
        if(pos<(direcao-10))
                {OCR2A=8;
                TCCR2A|=(1<<CS22)|(1<<CS21)|(1<<CS20);
                }
        else if(pos>(direcao+10))
                {OCR2A=14;
                TCCR2A|=(1<<CS22)|(1<<CS21)|(1<<CS20);
                }
        else if((pos>=(direcao-10))&&(pos<=(direcao+10)))
                {OCR2A=11;
                TCCR2A|=(1<<CS22)|(1<<CS21)|(1<<CS20);
                }
        }

int main(void){
        ad_ini();
        can_ini();

        //configuração da saída pwm
        TCCR0A = (1<<WGM00)|(1<<WGM01)|(1<<COM0A1); //Fast PWM, ZERA OC0A
NA COMPARAçÃO com TCNT0
        TCCR2A = (1<<WGM20)|(1<<WGM21)|(1<<COM2A1);//FAST PWM, ZERA OC2A
NA COMPARAçÃO com TCNT2
        DDRB=(1<<PB7)|(1<<PB4);  //ativa pinos de saida para sinal PWM

        sei();
        while(1){
        ad_conv(1,&aceleracao);
        ad_conv(4,&direcao);

        pos_ac(acel_rq);
        pos_dir(dir_rq);
        }
        return 0;
}
```

14. Referências bibliográficas

[1] GUIMARÃES, Alexandre A. Eletrônica Embarcada em Automóveis - Parte 1. Saber Eletrônica, n. 363, p. 40-43, Abril 2003.

[2] GUIMARÃES, Alexandre A. Eletrônica Embarcada em Automóveis - Parte 2. Saber Eletrônica, n. 364, p. 20-24, Maio 2003.

[3] BOSCH, Automotive Semiconductors and Sensors. Disponível em: http://www.semiconductors.bosch.de/en/20/can/1-about.asp. Acesso em: outubro/2008.

[4] BOSCH, CAN Specification 2.0B. Disponível em www.semiconductors.bosch.de/pdf/can2spec.pdf. Acesso em: outubro/2008.

[5] KVASER, CAN (Controller Area Network). Disponível em http://www.kvaser.com/. Acesso em outubro/2008

[6] ATMEL, Microcontrollers for CAN Networking. CD-ROM, Novembro /2006

[7] STMicroelectronics, VNH3SP30 Datasheet. Disponível em http://www.st.com/stonline/products/literature/ds/12688/vnh3sp30-e.pdf. Acesso em: outubro/2008

[8] NXP, LPC2129 datasheet. Disponível em http://www.nxp.com/acrobat/datasheets/LPC2109_2119_2129_5.pdf. Acesso em outubro/2008

[9] MICROCHIP, PIC18F4580 datasheet. Disponível em http://ww1.microchip.com/downloads/en/DeviceDoc/39637c.pdf. Acesso em outubro/2008

[10] ATMEL, AT90CAN128 datasheet. Disponível em http://www.atmel.com/dyn/resources/prod_documents/doc7679.pdf. Acesso em outubro/2008

[11] ATMEL, Efficient C Coding for AVR Application Note. Disponível em http://atmel.com/dyn/resources/prod_documents/doc1497.pdf. Acesso em outubro/2008

[12] AVR-LIBC, Reference Manual. Disponível em http://download.savannah.gnu.org/releases/avr-libc/. Acesso em outubro/2008

[13] ATMEL, Application notes AVR452 - Sensor-based Control of Three Phase Brushless DC Motors Using AT90CAN128/64/32. Disponível em http://www.atmel.com/dyn/resources/prod_documents/doc7616.pdf. Acesso em outubro/2008