Universidade do Porto
Faculdade de Engenharia
FEUP

Licenciatura em Engenharia Electrotécnica e de Computadores

Ramo APEL

Projecto Final de Curso 2000/01

SIMs

Sistema de Instrumentação Distribuído Multi-sensorial

Relatório Final

Julho 2001

Autores:
Gustavo Fernandes
Márcio Correia
Sérgio Almeida

Orientados pelo Prof. Dr. Adriano da Silva Carvalho

Apresentação

Este relatório trata da sistematização do trabalho por nós desenvolvido na disciplina de projecto, seminário ou trabalho final de curso da Licenciatura de Engenharia Electrotécnica e de Computadores, ramo APEL, realizado entre Fevereiro e Julho de 2001 nos laboratórios de projecto de APEL.

O projecto tinha por objectivo o desenvolvimento de um sistema de instrumentação distribuído baseado num protocolo de comunicação série.

Agradecemos todo apoio e ajuda prestada pelo nosso orientador Prof. Dr. Adriano da Silva Carvalho, pelo Prof. Dr. Manuel Barbosa, pelo Prof. Dr. Artur Cardoso, pelo Prof. João Paulo Sousa e por todos os restantes professores e colegas que nos acompanharam ao longo deste projecto.

Este relatório e demais informações por nós consideradas relevantes encontram-se na página Web do projecto alojada em www.fe.up.pt/~ee96112.

1. Motivação para o Projecto

O domínio de aplicação do nosso projecto é a monitorização e análise do comportamento de diversos sistemas de estruturas de Engenharia Civil e Mecânica, orientado pelo Professor Adriano Carvalho.

Quando escolhemos este projecto o principal atractivo foi a possibilidade de trabalhar em áreas um pouco esquecidas no ramo APEL da LEEC. Essas áreas são o projecto de sistemas digitais e a utilização de redes tanto para comunicação industrial, como no nosso caso para comunicação de instrumentação distribuída.

Assim, este projecto pareceu-nos ser a melhor forma de aprofundar o nosso conhecimento em áreas que pensamos ser de importância crucial na formação de um engenheiro do ramo APEL.

Numa primeira fase tivemos uma reunião com o Professor Adriano Carvalho, orientador do projecto, onde nos foi dado a conhecer quais os requisitos que este trabalho deveria alcançar, e as primeiras orientações sobre qual seria a melhor abordagem ao início do nosso projecto.

Após esta primeira reunião lançamos “mãos-à-obra” e começamos por fazer uma análise dos requisitos que o nosso sistema deveria comportar por forma a satisfazer os objectivos propostos no plano do projecto.

2. Análise de Requisitos

Nesta primeira fase, e após muitas discussões chegamos à análise de requisitos, isto é, o que é que nós pretendemos, de acordo com os objectivos propostos, que o nosso projecto incluísse.

O primeiro requisito foi a construção de um nó de instrumentação inteligente, através do qual poderíamos adquirir grandezas físicas diversas através de sensores e em seguida através de um nó distribuído, enviar essa mesma informação para um concentrador. Uma vez lá seria tratada e posteriormente enviada para a Web onde pudesse ser visualizada.

Com esta certeza acerca do primeiro ponto teríamos em seguida que ter um interface (rede) de comunicação série apropriada a grandes estruturas de Engenharia Civil e Mecânica, este interface serviria de suporte de comunicação entre o nó de instrumentação inteligente e o nó mestre ou concentrador.

O passo seguinte que centrou a nossa preocupação foi a definição da frequência de amostragem, assim desde logo foi especificado que a frequência de amostragem seria definida a nível de cada nó para que existisse liberdade no sistema, cada nó puderia estar a amostrar as grandezas que pretendesse sem qualquer inconveniente e qualquer confronto com outro nó. Para além deste factor ficou acordado igualmente que as características da amostragem seriam completamente independentes da dimensão da rede.

A nível da precisão entendemos ser razoável ter uma precisão melhor do que 0,005% na medição de grandezas diversas, ou seja, não nos interessava o sensor que ali pudesse ser colocado desde que as suas saídas fossem compatíveis com as entradas do conversor analógico-digital. Estas grandezas puderiam ser provenientes, por exemplo, de esforços mecânicos, temperatura, pressão, oscilações, etc. O objectivo era que o sistema (nó de instrumentação) fosse tanto quanto possível genérico de forma a poder ser adaptado a qualquer situação.

Com base nesta especificação de ser tão genérico quanto possível e de poder ser implementado em qualquer local pensamos numa capacidade para armazenamento de dados local e em seguida remota (concentrador) tal como uma capacidade de gestão da rede.

Para que todo este sistema funcionasse faltava somente a autonomia em funcionamento remoto para um mês, e a possibilidade de acesso remoto via Web.

Estes eram os requisitos que nós pensamos implementar no projecto. Com base neles passamos à fase seguinte a qual é a especificação do sistema.

3. Especificação do Sistema

Neste ponto e após a análise de requisitos começamos por uma pesquisa na Web e em livros sobre tudo o que pudesse interessar para que a análise de requisitos pudesse ser cumprida.

A quantidade de informação foi extremamente elevada e começaram aí as nossas dificuldades, pois a experiência era pouca e a informação em excesso, o que nos desviou de alguns objectivos.

A primeira grande dificuldade com que nos deparamos foi na escolha do A/D pois consoante a largura da banda dos sinais a amostrar, dos diversos tipos de sensores e da resolução que pretendemos para o sistema, deve-se escolher um A/D que verifique todos os requisitos atrás enumerados. A escolha recaiu sobre o AD7731 da Analog Devices por satisfazer todos os requisitos, e por já existir uma placa de desenvolvimento na sala de projectos de APEL. Mais à frente na escolha do hardware fazemos uma comparação entre este e outros A/D existentes no mercado (ver secção 7.5). Com este A/D temos uma solução para alguns dos requisitos enumerados. A frequência de amostragem fica optimizada para 800 Hz visto que as grandezas a medir são grandezas relacionadas com Engenharia Civil e Mecânica, que variam de uma forma lenta (≈ 100Hz), mas podem ir até ao máximo de 6,4 kHz. A precisão que podemos ter é melhor do que 16 bits a 800Hz, tendo este A/D uma resolução de 24 bits. Para além desta precisão tem ainda a possibilidade de aquisição em modo diferencial e pseudo-diferêncial, podendo o ganho e o filtro serem programáveis para a aquisição em cada A/D (nó de instrumentação).

Após a escolha do A/D, o passo seguinte foi a escolha do interface (rede de comunicação), também aqui as dificuldades foram enormes pois mesmo tendo já decidido de início usar o CAN como suporte, a pesquisa levou-nos a equacionar outras possibilidades. Assim, e após a comparação entre esta rede e outras (comparação que se apresenta na secção 4)

optamos pelo CAN, visto ser um protocolo mais robusto, não sujeito a erros e facilmente realizável.

Em seguida, a atenção recaiu sobre a forma como implementar o nó, ou seja, escolher um micro que nos desse a possibilidade de implementar o CAN. A primeira abordagem e talvez por ser a mais conhecida e tradicional, foi implementar uma solução com um microcontrolador e um controlador CAN em separado, mas após uma discussão mais aprofundada (ver secção 7.1) a escolha foi para o DS80C390 da Dallas Semiconductors que é um microcontrolador com controlador CAN incorporado.

Com a parte do nó já implementada faltava definir a parte do concentrador (PC), para que tal fosse possível necessitamos de um barramento de expansão para o PC de modo que tivesse a possibilidade de interface CAN. Com a ajuda de pesquisas na Web, a prioridade foi para a implementação de uma placa ISA com CAN incorporado, mas após algum tempo de investigação e algumas conversas com o Dr. Manuel Barbosa (ver referências bibliográficas), optamos por desistir desta ideia por não ser exequível em tempo útil (ver ponto 11.). Desta forma, após alguma pesquisa optamos por trabalhar com a placa de expansão para PC da National Instruments, PCI-CAN Dual Port CAN 2.0B.

Para além do CAN é possível comunicar com o micro através de RS-232 pois o micro implementa 2 portas série. Uma das portas série serve para comunicar com o A/D de forma a poder enviar as instruções para a sua configuração e para retornar o resultado da digitalização, a outra porta série vai ser utilizada para fazer o boot loader, ou seja, carregar o programa a partir do PC sem a necessidade de remover a EPROM do seu local. Outra vantagem é a possibilidade de efectuar a monitorização de registos via RS-232.

A última preocupação nesta fase foi a necessidade de alimentação do sistema remoto por forma que fosse totalmente autónomo. A forma como resolvemos o assunto foi pensando num sistema de alimentação com baterias recarregáveis através de painéis fotovoltáicos. As baterias seriam descarregadas na alimentação do nó distribuído, e carregadas através

de um pequeno sistema fotovoltáico. Desta forma garantíamos o funcionamento autónomo e com imunidade a flutuações de alimentação. Garantimos este ponto através da introdução de um regulador de tensão (Traco Power TEL 3-1211) que pode ser de alimentação variável entre 9 e 18 DC, proporcionando uma tensão de saída de 5V (alimentação da placa do nó distribuído e placa de instrumentação) com um máximo de corrente de saída de 600 mA (mais informação na secção 8).

Um outro vector que direccionou toda a especificação do projecto foi a redução de custos, ou seja a escolha de componentes e tecnologias que reduzam o custo final não nos despreocupando sempre de implementar uma solução robusta.

4. Escolha do protocolo a utilizar

4.1 Field Bus e seu Enquadramento

Os sistemas de comunicação fabril estão definidos segundo a ISO (International Standard Organization), em 6 partes:

- Empresa
- Fábrica
- Área
- Célula
- Estação
- Dispositivo

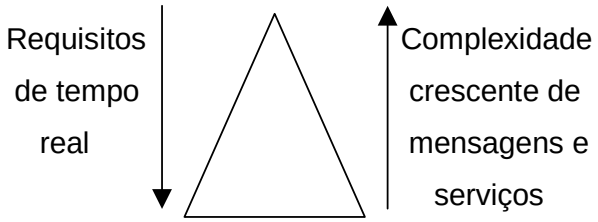

Fig. 1 – Sistemas de comunicação fabril

À medida que se desce na pirâmide aumentam os requisitos de tempo e a gestão de acesso ao meio torna-se mais apertada. Por outro lado a complexidade e agregação de informação intensifica-se à medida que caminhamos no sentido ascendente da pirâmide.

Como em cada nível as actividade são diferentes, as redes utilizadas devem satisfazer as necessidades de modo a poderem formar um conjunto operacional.

Nas redes de empresa/fábrica o objectivo a cumprir é, assegurar a informação relativa ao planeamento da produção de processos e materiais. O fluxo de informação presente nestas redes pode ter uma natureza técnica ou financeira.

Ao nível da área encontra-se a supervisão, controlo, monitorização e coordenação das actividades produtivas.

Nas Redes de Célula abordam-se as necessidades dos níveis intermédios, célula e estação. Estas encarregam-se ainda de funções como o escalonamento, execução de tarefas e sequenciamento.

As Redes de Campo cobrem as necessidade dos níveis mais baixos: estação e dispositivos. Destinam-se a interligar equipamento de controlo, sensores e actuadores. Neste campo encontram-se as redes de CAN, Bitbus, Profibus, de entre outras.

Visto que na aplicação desenvolvida utilizamos uma rede de campo apenas faremos exposição destas redes.

As redes de campo surgiram de modo a responder às necessidades de interligação de dispositivos de baixo nível, como já foi referido. Neste contexto convém explicar algumas das características que distinguem estas redes:

- O tipo de informação é reduzido;
- Podem ser aplicadas à industria de processos ou manufactura: tráfego aperiódico ou tráfego periódico;
- Os tempos de atraso são curtos;
- Utilizam meios redundantes;
- Os mecanismos de controlo de erro são sofisticados;
- O custo global é reduzido, com simplicidade da estrutura de suporte o maior possível;
- Apresentam segurança em relação a ambientes fabris perigosos;
- A largura da banda não é critica, salvo excepções.

Tal como em todas as tecnologias relacionadas com comunicação é preciso um longo processo para se poder falar num standard. Salienta-se o facto de muitas das redes deste nível serem redes proprietárias.

4.2 Profibus

É considerada por muitos como sendo a rede de campo mais popular. O seu aparecimento deu-se em 1989 na Alemanha, sendo os seus fundadores encontram-se ligados ao Governo Alemão e à Indústria Automóvel.

A implementação do protocolo é feita por intermédio de circuitos dedicados ASIC, produzidos por diversos fabricantes.

O interface físico é feito via RS-485. Presentemente existem várias especificações, algumas das quais são soluções proprietárias. A titulo informativo podemos citar o Profibus DP com arquitectura de Master/Slave, Profibus FMS com uma arquitectura multi-master e ponto a ponto, e por fim Profibus PA que dadas as suas características é intrinsecamente seguro. A nível de conectores estas redes utilizam fichas de 9 pinos, tipo DB-9 com resistências de terminação ou então conectores de 12 mm de ligação fácil.

O Profibus aceita no máximo 127 nós, sendo a distância total compreendida entre 100m a 24 Km, dependendo do tipo de meio utilizado e da utilização de repetidores.

Em termos de velocidades de transferência o Profibus aceita velocidades de 9600 bits/s a 12 Mbit/s. As tramas Profibus podem ser do tipo pooling ou do tipo ponto a ponto.

As aplicações típicas destas redes englobam controlo de processos, máquinas, armazenamento de materiais, válvulas pneumáticas e interfaces com operador tais como painéis tácteis, etc.

Conclusões

O Profibus é a rede de campo que mais é utilizada. O Profibus consegue transmitir grandes quantidades de informação com elevada velocidade. Como desvantagens do profibus podemos citar o enorme “over head” introduzido, e o facto de não existir a possibilidade dos nós serem alimentados pelo barramento.

4.3 Bitbus

Criado e desenvolvido pela Intel a partir de 1983, foi promovido a standard em 1990. Trata-se de uma rede do tipo Master/Slave em que cada mensagem tem um comprimento máximo de 248 bytes. A estrutura destas redes assenta num ou mais pares entrelaçados com a linha de massa, ligada à malha exterior. A impedância destas linhas é de 120 Ohm tal como noutras redes a fim de não ocorrem reflexões, utilizam-se resistências de terminação. Os níveis de tensão no Bitbus assentam nos mesmos valores do RS-485, ou seja no modo diferencial de 0-5V.

A nível de protocolo físico o Bitbus utiliza NRZI.

É possível atingir o número de 28 escravos por segmento, sendo o número máximo de segmentos de 250, contudo à medida que o número de segmentos aumenta, a taxa de transferência diminuí consideravelmente, para valores da ordem dos 60 Kbit/s.

No que respeita à extensão máxima o Bitbus define um máximo de 300 m por segmento a 375 Kbit/s e 1200 m a 62,5 Kbit/s com o auxílio de repetidores.

Em relação a conectores o Bitbus optou por conectores de 9 pinos sub-D.

Numa rede Bitbus existe, tal como já foi referido, a possibilidade de ligar 250 nós, cada nó é identificado por um endereço que deve ser único numa rede, a configuração de cada endereço pode ser feita por jumpers ou através de configurações de software. Os endereços compreendidos entre 250 e 254, assim como o endereço 0 são reservados, o endereço 255 é o endereço utilizado para difusão.

Com o intuito de simplificar o protocolo, o Bitbus define que apenas o mestre origina as mensagens e recebe as respostas dos nós escravos. Um nó escravo não pode transmitir sem antes ter sido solicitado por um mestre, deste modo o acesso ao meio está sempre garantido atendendo a que o nó mestre encarrega-se de indicar qual dos nós é que vai utilizar o meio.

Transmitida a mensagem do mestre, e a solicitação a um determinado escravo, este encarrega-se de transmitir a informação o mais depressa

possível. Sempre que um escravo demora mais tempo que o necessário o mestre envia uma nova mensagem de solicitação.

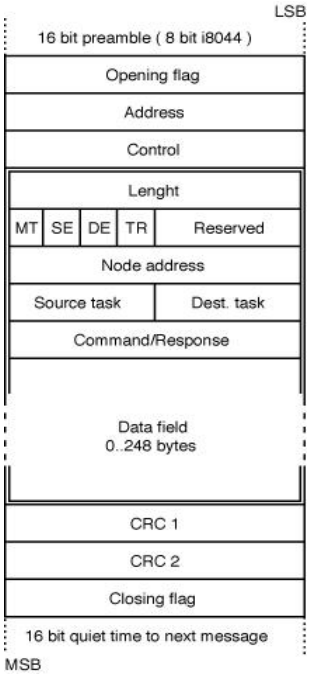

Fig. 2 – Trama Bitbus

Estrutura da trama do Bitbus:

- MT - Diferencia se é um pedido do mestre, MT=0, ou uma resposta de um slave, MT=1.
- SE - Indica que o processador do nó não é o criador da mensagem, mas sim outro processador que o controla.
- DE - Envia a mensagem para um processador que se encontra noutro sublanço de rede.
- TR - Flag de transmissão e recepção.
- NODE ADDRESS - Especifica o destino (escravo) 0...249

Resumo:

O Bitbus apresenta algumas características interessantes nomeadamente a possibilidade de arquitecturas mestre/escravo, utilização de

interfaces de acesso ao meio simples (RS-485). Como principal desvantagem pode-se focar o “over head” introduzido devido à possibilidade de escalonamento deste tipo de redes. Uma outra questão que se levanta quando se pretende utilizar este tipo de rede enquadra-se com a importância, ou não, de se ter uma arquitectura distribuída.

4.4 AS-I

Criada para ser a mais barata e simples rede de campo, esta rede oferece muitos dos benefícios de outras redes de campo mas com um custo relativamente inferior. Esta tecnologia está optimizada para a interligação de componentes digitais. Segundo a AS-Interface, organização que desenvolveu o AS-I, podem ser facilmente interligados sensores e actuadores convencionais a este tipo de redes.

Estas redes possibilitam a interligação de 31 nós escravos e um mestre, numa distância que pode atingir os 100 m ou 300 m (com o auxílio de repetidores), a velocidade de transferência é de 167 Kbit/s.

Uma grande vantagem destas redes é a interligação de dispositivos ser feita de um modo extremamente simples, os conectores encaixam nos condutores, conhecidos como cabos auto-cicatrizantes, assim chamados dada a capacidade de acrescentar e retirar componentes sem haver a necessidade de cortar estes cabos. No seguimento desta vantagem acrescenta-se que esta rede possibilita a alimentação dos dispositivos via cabos de ligação.

Tipicamente este tipo de redes é encontrado em blocos de entradas/saídas digitais, sensores inteligentes, válvulas pneumáticas, interruptores e sinalizadores.

Resumo:

O AS–I é uma rede de extrema simplicidade, excelente para dispositivos de I/O digitais. Como desvantagem desta rede aponta-se a dificuldade em interligar entradas e saídas analógicas.

4.5 Interbus

O Interbus apareceu em 1984 e intitula-se como sendo uma rede de campo de alta velocidade. Este tipo de redes possibilita a interligação de 256 nós distribuídos em segmentos com uma extensão máxima de 400 m, totalizando no máximo uma extensão de 12,8 Km.

O baud rate destas redes é de 500 Kbit/s. Como vantagens aponta-se a capacidade de auto-endereçamento o que possibilita configurações muito simples, baixos "over heads", pequenos tempos de resposta e uma eficiente utilização da largura de banda disponível na rede.

No que respeita a desvantagens indica-se a saída de operação de toda a rede quando a ligação é interrompida e a baixa capacidade para transferir grandes quantidades de informação.

4.6 Controlnet

A origem do Controlnet remete para a Allen-Bradley em 1995, este protocolo é baseado no "G6/U cabling" conhecido no meio das transmissões de TV por cabo. O número máximo de nós que este protocolo aceita é de 99, a extensão máxima que pode atingir é os 250 m ou 5000 m, com o auxílio de repetidores.

A taxa de transferência máxima é de 5 Mbit/s e o comprimento da mensagem pode variar de 0 a 510 Bytes. O formato das mensagens é baseado no modelo cliente/servidor o qual possibilita a existência de múltiplos mestres e ligações ponto a ponto. Outra característica importante deste protocolo é a capacidade de encaminhamento das mensagens por caminhos alternativos, com esta característica este protocolo apresenta uma enorme redundância a erros.

As aplicações típicas deste protocolo centram-se na interligação de PC's a sub-redes de PLC´s, controlo de processos, situações em geral que requerem um grande volume de informação e em simultâneo o factor tempo seja crítico.

Como mais valia deste protocolo pode-se citar que este é um protocolo determinístico, usa de forma eficiente a largura de banda disponível, oferece redundância a baixos custos (isto quando comparado com outras redes). Pode ser implementado utilizando vários tipos de meios como a Ethernet, Firewire ou USB.

Como desvantagens aponta-se que poucos vendedores o adoptaram devido ao elevado preço dos controladores fabricados pela Rockwell.

4.7 FIP

O FIP (Factory Instrumentation Protocol) é uma rede desenvolvida a partir de meados dos anos 80 em França. Os seus criadores encontravam-se ligados à Telemecanique e a diversos centros de investigação. Presentemente o FIP é um standard da industria Francesa.

Segundo os seus impulsionadores o FIP é uma rede de campo vocacionada para a interligação de dispositivos de baixo nível tais como sensores, actuadores e autómatos.

A utilização do FIP é bastante flexível, a velocidade de transferência pode chegar aos 5 Mbit/s e o comprimento global pode atingir os 4500 m.

5. O Modelo OSI

Este modelo descreve como as comunicações devem ocorrer entre computadores numa rede, e foi adaptado como um standard nas comunicações em redes. Em principio tudo o que esteja de acordo com o standard pode comunicar electronicamente com qualquer outro dispositivo que esteja igualmente em conformidade com o modelo.

O modelo OSI define 7 níveis diferentes que devem existir num protocolo, este níveis devem estar bem isolados uns dos outros por interfaces bem definidos. No entanto, nenhuma rede está em total conformidade com este modelo em que os 7 níveis estão bem definidos.

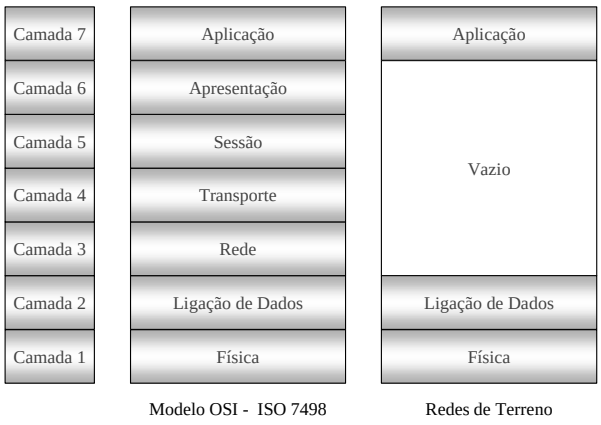

Fig. 3 - Modelo OSI

5.1 O Modelo OSI e o CAN

O protocolo CAN apenas usa três das sete camadas definidas pelo modelo OSI: a camada física, a camada de ligação de dados e a camada de aplicação.

O nível físico especifica as características físicas e eléctricas do barramento, e como deve o hardware converter os caracteres de uma mensagem em sinal eléctrico para poderem ser transmitidas e vice-versa para poderem ser recebidas. Enquanto todas as outras camadas podem ser

implementadas tanto em hardware como em software, a camada física é a única que só pode ser implementada a nível do hardware. O CAN possui uma topologia em bus e utiliza um par entrelaçado como meio de transmissão. O sinal eléctrico é transmitido em modo diferencial (idêntico ao RS – 485) e permite que em caso de interrupção de um dos condutores, a transmissão ainda seja possível.

Os dados transmitidos são codificados segundo o código NRZ, produzindo este menor interferência electromagnética a velocidades elevadas do que os restantes códigos. A velocidade de transmissão máxima é de 1 Mbps para distâncias não superiores a 50 metros, para distâncias superiores a velocidade de transmissão decresce proporcionalmente com a distância.

A camada de ligação de dados é a única camada que reconhece e entende o formato das mensagens. Este nível constrói as mensagens a serem enviadas pela camada física e descodifica as mensagens recebidas pela mesma camada física. Geralmente nos controladores CAN a camada de ligação de dados é implementada em hardware.

O Porquê da Existência do Vazio

O vazio que aparece no modelo em camadas para o CAN (e em geral para todas as redes de campo) fica explicado se focarmos o objectivo de cada camada desse vazio.

A camada de rede é responsável pelo envio da trama desde a origem ao destino, ou seja, trata do encaminhamento, controlo e congestionamento e mesmo da interligação de redes diferentes. Como o CAN é uma rede única esta camada não é necessária.

A camada de transporte está mais virada para fornecer um serviço fiável end-to-end ou seja entre utilizadores e não entre nós numa rede. Tem funções semelhantes à camada de ligação lógica mas mais virada para a totalidade da rede. Fornece as funções para satisfazer a qualidade de serviço requerido pelo nível superior em termos de taxas de transmissão, atrasos, etc. Mais vocacionado para protocolos de alto nível.

A camada de sessão tem a ver com a sincronização de secções de diálogo distribuídas. Normalmente esta camada não é usada devido à sua especificidade.

A camada de apresentação trata da compatibilidade de formatos dos dados. Pode também tratar da compressão de dados ou da sua encriptagem. Esta camada normalmente está mais presente na camada de aplicação.

Resumindo, as comunicações locais industriais deste nível envolvem dispositivos ligados ao mesmo meio físico, mensagem de pequena dimensão e necessidade de conversão de formatos reduzida. Desta forma explica-se o vazio do CAN relativamente ao modelo OSI.

O CAN não possui camada de aplicação, contudo foram desenvolvidas por algumas companhias pacotes de software que implementam serviços da camada de aplicação. Alguns destes softwares de aplicação são: CanKingdom, CanOpen/CAL, DeviceNet, SDS.

6. CAN (Controller Area Network)

O CAN foi desenvolvido no início dos anos 80 pela Bosch, para aplicações na indústria automóvel. O objectivo era permitir a interligação de dispositivos electrónicos (p. ex. controlo do funcionamento do motor, ABS, controlo de suspensão, controlo de luzes, airbags, etc.) através de uma solução de baixo custo, e que garanta uma redução importante nas cablagens.

Além do CAN, foram desenvolvidos outros protocolos para o uso na indústria automóvel como o ABUS da Volkswagen, o VAN da Peugeot e Renault e o J1850 da Chrysler, General Motors e Ford. Estes protocolos diferem fundamentalmente ao nível da taxa de transferência, código, formato das mensagens, detecção de erros e seu tratamento.

Actualmente o CAN tem vindo a assumir uma posição de líder do mercado, e muitos construtores têm vindo a abandonar os seus protocolos, proprietários, adoptando o CAN. Na prática verificou-se que devido às suas características poderia ser utilizado como uma rede à qual se podem interligar sensores/actuadores ou outros dispositivos, baseados em microcontroladores/microcomputadores.

A robustez e eficiência do CAN, aliada à disponibilidade de circuitos integrados dedicados a este protocolo (actualmente estão disponíveis mais de 50 chips com controlador CAN de cerca de 15 fabricantes), ditou o alargamento do seu uso à automação industrial, equipamentos médicos, navios, domótica, robôs, CNCs, e entre outras, a aplicações de controlo e monitorização de processos sobre o qual vai incidir todo o nosso projecto.

O CAN é um protocolo standard ISO (ISO 11898) para comunicação de dados série com capacidade multi-master, isto é, todos os nós CAN têm a capacidade para transmitir, e alguns podem fazer pedidos à rede simultaneamente.

Teoricamente uma rede CAN pode estar ligada a 2032 nós (assumindo 1 nó com um identificador). No entanto, devido a limitações práticas de

hardware (transceivers), uma rede só pode estar ligada a 110 nós (com o tranceiver da Philips 82C250).

6.1 As Mensagens CAN

O CAN é um barramento do tipo broadcast, isto quer dizer que todos os nós ouvem todas as transmissões. Não existe forma de enviar uma mensagem para um determinado nó, pois todos os nós ouvem essa mensagem. O hardware do CAN, no entanto, pode conter filtros locais de forma que cada nó somente reaja a mensagens que têm interesse.

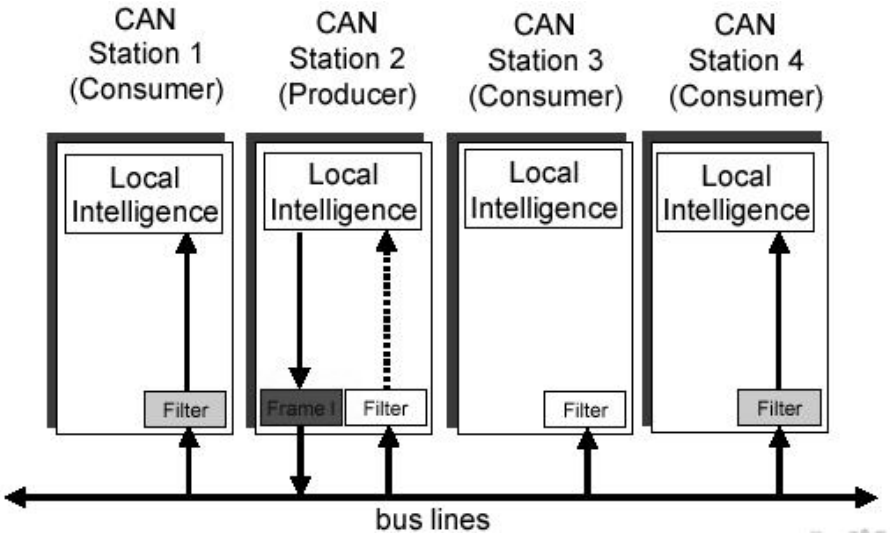

Fig. 4 – Transmissão em broadcast e filtros de recepção

O CAN usa pequenas mensagens – no máximo de 94 bits.

6.1.1 Tipos de Mensagens

Existem quatro tipos de tramas (frames) na comunicação CAN:

- Trama de Dados (Data Frame)
- Trama Remota (Remote Frame)
- Trama de Erro (Error Frame)
- Trama de Sobrecarga (Overload Frame)

Trama de Dados (Data Frame)

A trama de dados é o tipo mais comum de mensagem. Esta trama é composta pelos seguintes 7 campos:

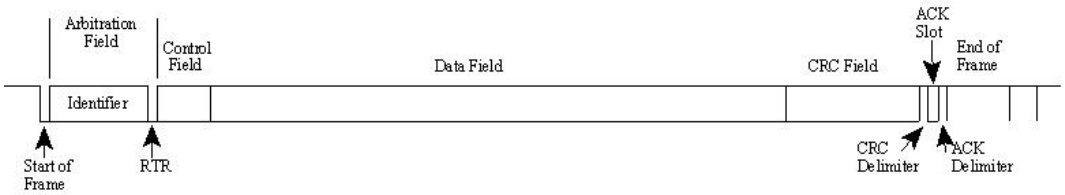

Fig. 5 – Trama de dados CAN2.0A (Trama Standard)

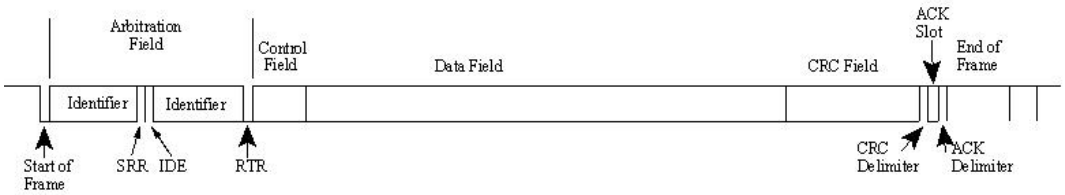

Fig. 6 – Trama de dados CAN2.0B (Trama Estendida)

- Bit de Início de Trama (Start of Frame) – é composto por um bit dominante (“0”), que assinala o início de uma trama de dados ou de uma trama remota e força a sincronização de um controlador CAN no modo de recepção. A partir deste momento todos os restantes nós terão de se sincronizar por ele e começar a escutar, ou a competir por ele.

- Campo de Arbitragem (Arbitration Field) – determina a prioridade de uma mensagem quando dois ou mais nós estão a competir pelo meio. O campo de arbitragem tem um identificador com 11 bits para o CAN 2.0A, mais 1 bit, o RTR, que é dominante para as tramas de dados. Para o CAN 2.0B, o identificador tem 29 bits (e contem igualmente mais dois bits recessivos: o SSR e o IDE) para além do bit RTR. O bit RTR indica se a trama transmitida é uma trama de dados (bit a 0) ou a uma trama remota (bit a 1).

As tramas com identificadores standard têm prioridade mais alta do que as tramas com identificador estendido.

Esta arbitragem é feita como se mostra na figura seguinte:

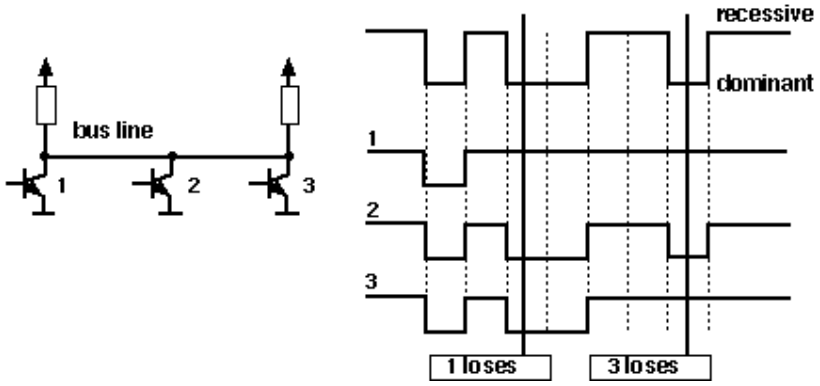

Fig. 7 – Processo de arbitragem entre três nós

Quando um nó envia um SOF para a rede todos os outros se sincronizam com ele, e quem quiser competir pelo meio envia o seu identificador. Ao realizarem este procedimento, além de enviarem a sua identificação estão ao mesmo tempo a escutar o meio. Este escutar do meio serve para confirmar o que estão a enviar. Como os nós estão ligados em wired-and quando um bit dominante é enviado para o meio por qualquer nó, todos os bits recessivos que outros enviam são sobrepostos. Quando detectam que existe uma diferença entre o que enviam e o que está no meio passam instantaneamente ao estado de escuta, retirando-se assim da competição pelo meio.

A competição pelo meio é do tipo CSMA/CD (Carrier Sense, Multiple Acess / Collision Detection), ou seja competição pelo meio não destrutiva. Ao chegar ao fim do campo de arbitragem quem tiver ganho a competição fica com o meio e pode continuar a transmissão, ficando todos os outros como ouvintes.

- Campo de Controlo (Control Field) – é composto por 6 bits, 2 bits reservados e o código de comprimento de dados (DLC, Data

Length Code), que indica o número de bytes de dados (sem bit stuffing) a serem transmitidos / recebidos no campo de dados. Os valores admissíveis para o DLC vão de 0 a 8 bytes.

- Campo de Dados (Data Field) – os dados guardados no campo de dados do buffer de transmissão são transmitidos de acordo com o código de comprimento de dados. Inversamente, os dados recebidos através da trama de dados são guardados no campo de dados do buffer de recepção. O campo de dados pode ter de 0 a 8 bytes.

- Código de Redundância Cíclica (CRC, Cyclic Redundancy Code) – tem 15 bits para a sequência CRC e 1 bit recessivo ("1") para o delimitador de CRC. O Código de redundância cíclica é o quociente da divisão por (2^{15}-1) da soma das sequências de bit (sem bit stuffing) do bit de inicio de trama, do campo de arbitragem, do campo de controlo e do campo de dados. O delimitador de CRC serve para delimitar o campo anterior e para detectar a ocorrência de eventuais colisões no barramento.

- Campo de Confirmação (Acknowledge Field) – é composto por 3 bits, o intervalo de confirmação (acknowledge slot) é o delimitador de confirmação (acknowledge delimiter), que são transmitidos com um nível recessivo pelo transmissor da trama de dados. Quando um controlador CAN recebe a sequência de CRC correspondente, sobrepõe um bit dominante ao bit recessivo do intervalo de confirmação. O delimitador de confirmação é um bit recessivo e serve para delimitar o campo anterior.

- Fim de Trama (End of Frame) – cada trama de dados ou trama remota é delimitada por este campo, que é composto por 7 bits recessivos, o que excede a largura de bit stuffing por 2 bits. Ao usar

este método, o receptor detecta o fim de trama independentemente de existirem ou não erros, pois o receptor espera que todos os bits, até ao fim da sequência de CRC, estejam codificados pelo método de bit stuffing.

Trama Remota (Remote Frame)

Este tipo de trama é em tudo igual à trama de dados, excepto em dois pontos. O primeiro é que o bit do campo de controlo RTR (Remote Transmission Request) é igual a um para indicar este tipo de trama. O segundo é que esta trama não contém campo de dados. Apesar deste aspecto, o campo DLC pode tomar qualquer dos valores possíveis. Para evitar erros, se mais do que um nó enviar uma trama remota com o mesmo identificador, o campo DLC deve ter um valor fixo, como por exemplo, o valor do DLC pode ter o valor dos dados pedidos.

Normalmente este tipo de trama serve para um nó remoto pedir uma variável remota que pertence a outro nó na rede CAN.

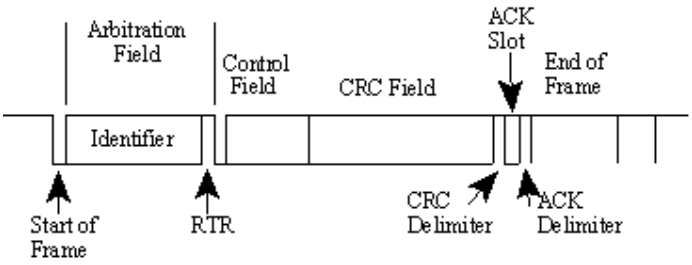

Fig. 8 – Trama remota (RTR)

Trama de Erro (Error Frame)

Este tipo de trama serve para indicar que uma trama enviada para o meio chegou com erro a um nó e é composta por dois campos:

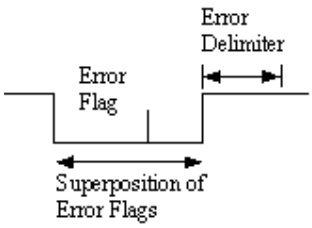

Fig. 9 – Trama de erro

- Sobreposição de Flags de Erro (Superinposing of Error Flags) – existem dois tipos de flag de erro:

  - Flag de erro activa que consiste em 6 bits dominantes consecutivos e que é transmitida por um controlador CAN activo para erros;

  - Flag de erro passiva, que consiste em 6 bits recessivos consecutivos, a não ser que sejam sobrepostos por bits dominantes de outros controladores. Esta flag é transmitida por um controlador CAN passivo para erros.

Quando um controlador CAN, activo para erros, detecta um erro, transmite uma flag de erro activa. Como esta flag viola a regra de bit stuffing, ou viola a forma fixa dos campos a serem transmitidos no momento, os outros controladores detectam um erro e começam a transmitir as suas flags de erro. Deste modo, a sequência de bits que se encontra no barramento é composta pela sobreposição de várias flags de erro e pode ter no mínimo 6 e no máximo 12 bits.

- Delimitador de Erro (Error Delimiter) – é composto por 8 bits recessivos. Depois de terem transmitido a flag de erro, os controladores CAN escutam o barramento até detectarem uma transição de dominante para recessivo. Neste momento, já todos os controladores acabaram de enviar a flag de erro e já enviaram os primeiros 3 bits do delimitador de erro. Depois de enviar o resto do

delimitador e após um campo de intervalo, todos os controladores CAN podem reiniciar a transmissão.

Se o erro foi detectado durante a transmissão de uma trama remota ou de uma trama de dados, a mensagem fica danificada e tem de ser retransmitida.

Se um controlador CAN detecta um desvio na trama de erro, transmite uma nova trama de erro. Várias tramas de erro consecutivas podem colocar o controlador no estado passivo para erros.

Trama de Sobrecarga (Overload Frame)

Este tipo de trama é idêntica à de erro, ou seja, é constituído por 6 bits recessivos mais o delimitador.

Esta trama é enviada nas seguintes situações:

- quando um receptor não está preparado para aceitar uma mensagem e carece de mais tempo para se preparar;
- se detectado um nível dominante no campo de intervalo entra tramas;
- ou se detectado um nível dominante no oitavo e último bit do delimitador de erro ou de sobrecarga.

A transmissão de uma trama de sobrecarga só pode ser iniciada nas seguintes condições:

- durante o primeiro período de bit do campo de intervalo;
- um período de bit após ter detectado um bit dominante durante o campo de intervalo.

Esta trama é tratada de forma diferente da trama de erro, pois a trama de sobrecarga não obriga à retransmissão da trama anterior.

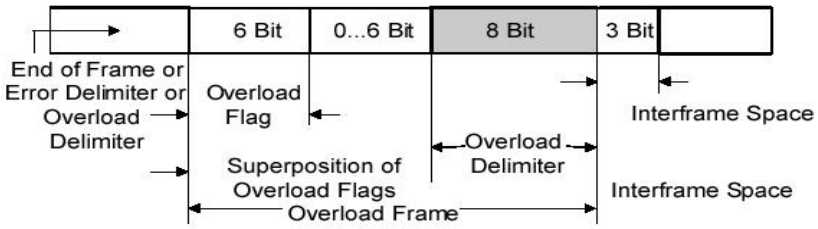

Fig. 10 – Trama de overload e espaço entre tramas

Espaço entre Tramas (Interframe)

O espaço entre tramas separa as tramas do tipo dados ou remota de qualquer outro tipo que as precedam. Isto não acontece com os outros tipos de tramas: erro e de sobrecarga.

Este espaço é constituído por três partes:

- intermissão;
- transmissão suspensa;
- barramento inactivo.

A primeira é constituída por 3 bits recessivos. A seguinte por 8 bits recessivos, o envio desta secção só acontece quando um nó erro-passivo acaba de transmitir uma trama. Assim, este nó só reconhece o barramento como livre mais tarde, dando oportunidade de nós com o estado de erro anterior, ou seja, erro passivo, possam transmitir.

A última é de tamanho indeterminado e onde qualquer nó detecta o barramento como livre. Por consequência qualquer bit dominante é aqui entendido como um SOF.

6.2 Conformidade entre versões (Standard Vs. Estendida)

Originalmente o standard CAN definia um identificador no campo de arbitragem com 11 bits. Mais tarde, e devido a uma exigência dos clientes foi pedida uma extensão do standard. O novo formato é designado por estendido (“Extended CAN”) e tem como identificador 29 bits. Para

diferenciar estes dois tipos de identificadores é usado um bit especifico do campo CRC.

Os standards são normalmente designados por:

- 2.0A, com um identificador de 11 bits;
- 2.0B, versão estendida com 29 bits no identificador (pode ter igualmente 11 bits que estão misturados). Um nó 2.0B pode ser:

  - 2.0B activo, isto é, pode transmitir e receber tramas estendidas;
  - 2.0B passivo, isto é, ignoram tramas recebidas com identificador estendido.

Os novos controladores CAN são normalmente do tipo 2.0B. Assim, um controlador CAN do tipo 2.0A se receber uma trama estendida não a reconhece dando origem a erro. Um controlador do tipo 2.0B passivo vai receber essa mesma trama e em seguida ignora o seu aparecimento.

Controladores 2.0B e 2.0A são compatíveis e podem usar-se no mesmo barramento, desde que o controlador CAN 2.0B não envie tramas estendidas.

6.3 Filtragem de Mensagens

A filtragem permite que o µc (microcontrolador) não seja interrompido constantemente aquando da chegada de uma mensagem, mas apenas quando elas lhe interessam. Esta filtragem centra-se no identificador, com uso opcional de uma máscara pode-se filtrar os bits do identificador que se quer.

6.4 Codificação do Fluxo de Bits

As secções começo de trama, campo de arbitragem, campo de controlo, campo de dados, e CRC são codificadas pelo método de bit stuffing. Assim, sempre que o transmissor da mensagem encontra cinco bits consecutivos do mesmo nível, insere um bit de valor complementar na mensagem transmitida. Todos os receptores farão o oposto para ficarem com a mensagem original.

Os restantes campos: delimitador de campo CRC, campo de acknowledge e o de fim de trama não serão codificados. Como é de esperar as tramas de erro e de sobrecarga, devido à intencionalidade, violam a regra do bit stuffing, pelo que não serão codificadas por esta regra.

A codificação do fluxo de bits é feita pelo método NRZ (Non Return to Zero).

6.5 Detecção e Correcção de Erros

Para detecção de erros, o protocolo CAN implementa 3 mecanismos ao nível da mensagem:

- Cyclic Redundancy Check (CRC) – Erro de CRC – Quando o cálculo do CRC por parte do receptor não iguala o valor do campo CRC recebido. Este cálculo do CRC por parte do receptor é feito de igual modo ao do transmissor da mensagem;

- Frame Check (Erro de Formato) – este mecanismo verifica a estrutura da trama de transmissão comparando os campos com uma de formato fixo. Os erros detectados são chamados erros de formato;

- Ack Error (Erro de Acknowledgement) – se o transmissor da mensagem não detectar um bit dominante na slot de acknowledge significa que nenhum nó na rede confirmou a recepção da

mensagem enviada. Esta situação provoca um erro de acknowledge.

O protocolo CAN implementa igualmente 2 mecanismos para detecção de erros ao nível do bit:

- Bit Monitoring – Erro de Bit – se o nível do bit monitorizado no barramento for diferente do enviado, é sinalizado um erro de bit. As excepções a esta regra são o campo de arbitragem, onde esta situação significa que o controlador não tem a mensagem de maior prioridade, e o espaço de confirmação, onde só os receptores podem detectar este erro;

- Bit Stuffing – verifica se o campo de inicio de trama, arbitragem, controlo, dados e CRC obedecem às regras de bit stuffing. Quando um controlador CAN detecta um erro transmite uma flag de erro. Esta flag é transmitida após o delimitador de confirmação, se for um erro de CRC, ou no bit seguinte ao erro, em todos os outros tipos. A flag de erro viola o bit stuffing, de modo que todos os controladores detectam o erro e iniciam igualmente a transmissão de uma flag de erro.

6.6 Estado de Erro de um Nó

Cada nó CAN tem dois contadores de erros: o contador de erros de transmissão e o contador de erros de recepção. Num nó numa rede CAN consoante o valor dos seus contadores de erro pode assumir um dos 3 estados:

- Erro Activo: o nó participa normalmente nos procedimentos da rede enviando uma trama de erro-activa sempre que detecte um erro;

- Erro Passivo: neste estado o nó substitui a trama erro-activo pela de erro-passivo, passando a existir uma maior desconfiança que provavelmente esse erro é atribuível a ele próprio;

- Bus Off: ao entrar neste estado o nó retira-se de toda a actividade do barramento, ou seja, desactiva os seus drivers de saída. Para regressar às actividades do meio tem que existir uma reinicialização do sistema ou em opção o aparecimento de 128 vezes 11 bits recessivos consecutivos. Assim os seus contadores, TEC e REC, voltam a zero e o nó ao estado de erro-activo e à normal actividade com o meio.

Para o controlo destes estados temos dois contadores:

- TEC (Transmitter Error Counter) ou contador de erros de transmissão;
- REC (Receiver Error Counter) ou contador de erros de recepção.

Estes contadores são incrementados ou decrementados segundo determinadas regras. Os incrementos podem ser de 1 ou de 8 consoante a gravidade do erro. O decremento faz-se quando uma transmissão/recepção é bem sucedida, sendo o seu valor de uma unidade.

O diagrama de blocos que implementa a passagem encontra-se descrito em seguida.

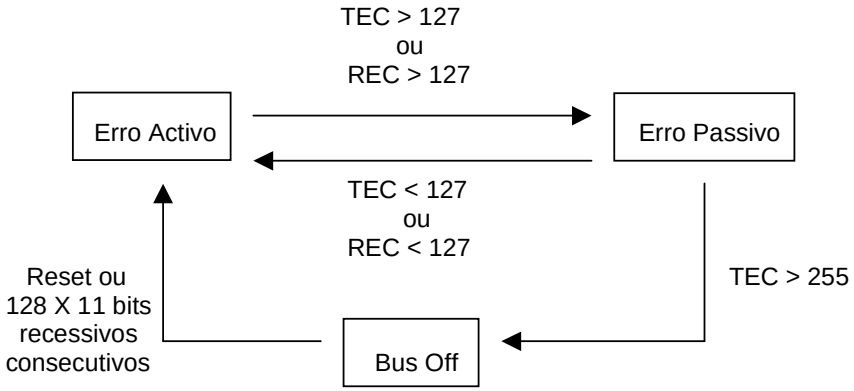

Fig. 11 – Diagrama de blocos do estado de erro de um nó

6.7 Temporizações de um Bit

Tempo Nominal de um Bit

Um bit é constituído por quatro secções que no seu conjunto formam o chamado Tempo Nominal de Bit. Estes componentes são:

- Segmento de Sincronização – normalmente tem a duração de 1 TQ (Time Quantum) e é usado para a sincronização dos clocks

- Segmento de Propagação – essencialmente serve para dar tempo à propagação do sinal no meio e pode ir desde 1 TQ a 8 TQ

- Segmento Phase Buffer 1 – este segmento e o seguinte servem para compensar erros de fase podendo, por resincronização, serem encurtados ou estendidos, pode ir de 1 a 8 TQ

- Segmento Phase Buffer 2 – o valor deste segmento é o máximo entre a secção anterior e o tempo de processamento da informação

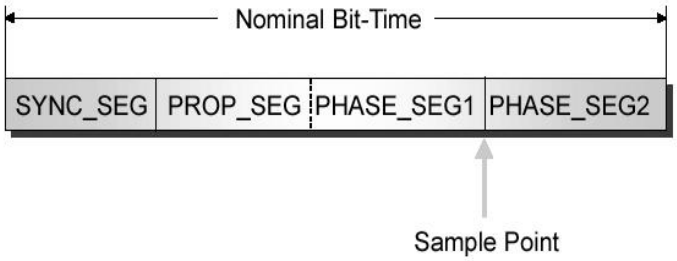

Fig. 12 – Tempo nominal de bit

O ponto de amostragem situa-se no final do segmento Phase Buffer 1 e é o ponto onde realmente deve ser lido o valor do nível do bit para determinar se é um bit recessivo ou dominante.

O tempo de um bit é dividido em intervalos iguais, esses intervalos são chamados Time Quantum (TQ) e são fornecidos pelos osciladores dos nós da rede, directamente ou através de um prescaler. Assim, um TQ é uma unidade fixa de tempo.

Atrás falou-se que os segmentos Phase Buffer 1 e Phase Buffer 2 podiam ser alongados ou abreviados através de resincronização. O n.º de TQs que podem ser acrescentados/retirados é o chamado Resynchronization Jump Whith (RJW ou SJW) que tem o mínimo de 1 TQ e o máximo entre o mínimo de 4 TQ e o segmento Phase Buffer 1.

6.8 Sincronização

O encurtar ou alargar dos segmentos de fase é a chamada Soft Synchronization e depende do estado da fase nesse momento. Para este tipo de sincronização são usadas as transições de bits recessivos para dominantes, e em opção de dominantes para recessivos no caso de Baud Rates baixos. Devido ao bit stuffing esta sincronização está garantida no máximo, cada 10 bits, ou 5 bits em opção para os baixos Baud Rates.

Uma outra forma de sincronização é a Hard Synchronization, que é feita sempre que é detectado um SOF (inicio de trama). Isto significa que o temporizador interno de bit será reinicializado.

6.9 Camada Física

Este nível pode ser dividido em 3 sub camadas:

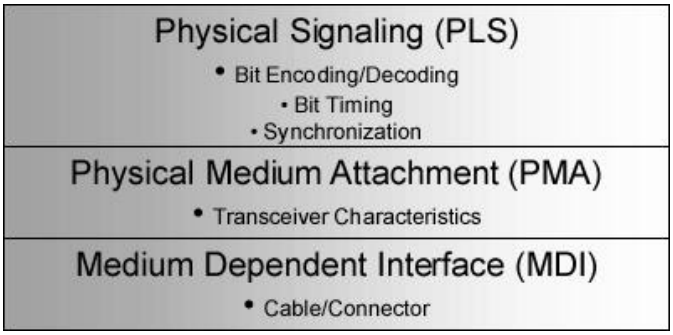

Fig. 13 – Camada física do CAN

A camada PMA é a última camada a ser implementada pelo controlador, as camadas seguintes são implementadas pelos transceivers e pelo meio físico.

A codificação utilizada pelo CAN segue o método NRZ (Non – Return – Zero). Isto implica que durante a transmissão de um bit o nível de tensão mantêm-se constante. Uma das características do NRZ é que esta codificação não inverte a polaridade e como tal não é possível fazer a resincronização quando se transmite uma grande quantidade de bits iguais. Para se evitar erros utiliza-se uma técnica de “Bit – Stuffing”.

A norma ISO 11898-2 assume que a ligação física é realizada por intermédio de um barramento único a fim de se diminuir as reflexões. Os barramentos CAN devem ser terminados nas duas extremidades com resistências de 120 Ω.

Níveis lógicos:

Os níveis de tensões detectados pelos transceivers indicam o estado do bit presente num determinado momento no barramento. Os transceivers detectam um nível recessivo se a tensão presente em CAN-H não for superior à tensão presente em CAN-L pelo menos em 0,5 V. Se a tensão em

CAN-H for pelo menos 0,9 V superior à tensão presente em CAN-L uma tensão de bit dominante é detectada. A tensão nominal no estado dominante é de 3,5 V para CAN-H e 1,5 V para CAN-L.

Dado que a natureza da transmissão é diferencial a tecnologia CAN é insensível a interferências electromagnéticas, isto porque ambas as linhas estão sujeitas ao mesmo ruído, logo o sinal diferencial mantêm-se inalterado.

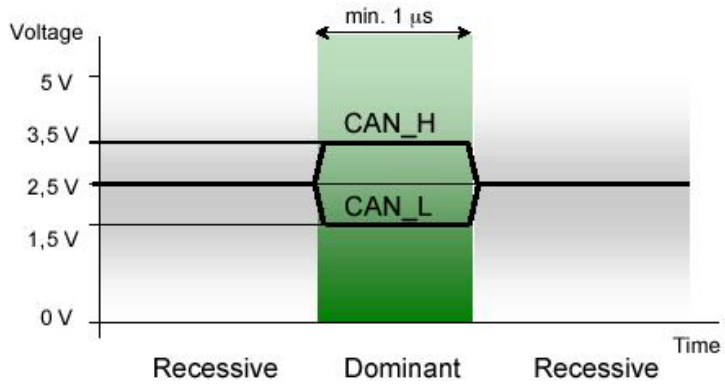

Fig. 14 – Níveis lógicos do CAN

Conectores a serem utilizados:

Os conectores a serem utilizados numa rede de CAN devem ser do tipo DB-9, em alternativa é possível utilizar outros conectores tais como “flat cables”, RJ10, RJ45, 5 pinos circular mini Din, etc.

6.10 Camada de aplicação do CAN

6.10.1 CanKingdom

O CanKingdom foi essencialmente desenvolvido para controlo de máquinas (p. ex. robôs industriais, teares máquina, etc.) e especificado pela firma sueca Kvaser. O CanKingdom suporta, além de outros aspectos: trocas de informação segura, alteração dinâmica de identificadores, um relógio geral, identificadores standard e estendidos. A desvantagem é a limitação de hardware dos módulos num sistema.

Num sistema CanKingdom, o módulo serve o sistema, isto é, não é necessário um conhecimento do sistema. Aqui, o “King” (nó supervisor) toma toda a responsabilidade pelo sistema, por exemplo, este nó supervisor especifica qual o nó que deve ser adicionado e sob que circunstâncias. Neste sistema de aplicação existe uma identificação simples e única para cada nó, cada nó tem um número de série que o identifica. O identificador aqui não só identifica a mensagem como também administra o acesso ao meio. Outro factor importante é que a estrutura dos dados, no campo de dados, é a mesma para os módulos de transmissão e de recepção. Se todos estes factores forem controlados a comunicação pode ser optimizada para qualquer sistema.

6.10.2 CAL (Can Application Layer)

Foi publicado em 1993 pela CiA (CAN-in Automation). Oferece uma aplicação independente, orientada ao objecto para a implementação de sistemas distribuídos baseados em CAN. Proporciona objectos e serviços para a comunicação e distribuição de identificadores. Muitas das aplicações baseadas em CAN não requerem configuração e estandardização dos dispositivos. Um subconjunto do CAL é utilizado como camada de aplicação do CanOpen.

6.10.3 CanOpen

O CanOpen é um protocolo de alto nível baseado na comunicação série CAN. É um standard especificado pela CiA (CAN – in Automation). A modelização do meio é baseada na descrição das suas funcionalidades, esta forma de descrição é também usada por outros protocolos como seja o Interbus-S e o Profibus.

O perfil (“profile”) será como um molde segundo o qual cada fabricante concebe o seu dispositivo. Um perfil não é mais do que um conjunto de funcionalidades de caracter obrigatório que vão garantir a operacionalidade e

interconectividade dos dispositivos enquanto membros intervenientes na rede.

As funcionalidades implementadas por um dispositivo CanOpen estão ordenadas segundo uma ordem predeterminada num dicionário de objectos acessível através da rede. Cada objecto do dicionário é acedido por intermédio de um índice de 16 bits e um sub-índice de 8. Aqui é feita a distinção entre as funcionalidades de comunicação comuns a todos os dispositivos e obrigatórias, e as especificas de cada dispositivo.

O CanOpen implementa um mecanismo de sincronização que suporta o funcionamento síncrono dos dispositivos, assim como a transmissão assíncrona de dados. Para o efeito são definidos diversos objectos de comunicação:

COB (Comunication Object)

Unidade de transporte de dados através da rede

SDO (Service Data Object)

Permitem aceder ao dicionário de objectos do dispositivo em comunicação ponto a ponto. Quando a informação a transferir excede os 8 bytes de dados esta é repartida e enviada em vários SDOs. A transferência de SDOs normalmente tem baixa prioridade.

PDO (Process Data Objects)

Vocacionado para a troca de dados entre vários dispositivos, “broadcast”. Os dados são enviados sem formatação especifica sendo o seu comprimento máximo limitado a 8 bytes.

LMT (Layer Management)

É um serviço do CAL que permite configurar os diversos parâmetros das camadas definidas pelo modelo de referência do CAN.

NMT (Network Management)

Mais um dos serviços oferecidos pelo CAL, através do qual se processam todos os mecanismos de inicialização, configuração, monitorização e tratamento de erros de uma forma standard.

DBT (Distributor)

A norma CAL define 3 métodos de distribuição de endereços:

- identificadores pré definidos;
- ferramenta de configuração;
- distribuição dinâmica e transparente oferecida pelo serviço DBT.

Cada aplicação pode definir o seu método de atribuição de identificadores.

6.10.4 DeviceNet

DeviceNet é uma rede aberta desenvolvida pela Allen-Bradley e mantida pela ODVA (Open DeviceNet Vendors Association). A sua principal aplicação é ligar dispositivos industriais (ex: Interruptores, sensores fotoeléctricos, válvulas, arrancadores, sensores de processo, leitores de código de barras, variadores de velocidade, displays de painéis e interfaces de operador) numa rede aberta onde todos os módulos (nós) têm o mesmo direito de acesso ao meio e este acesso é apenas regulado por poucas regras.

Define um perfil “Predifined Master / Slave Conection Set”, com características muito idênticas ao SDS.

6.10.5 SDS (Smart Distributed Systems)

SDS é um protocolo aberto especificado pela Honeywell, foi especificamente pensado para sensores inteligentes e actuadores directamente ligados a uma rede. SDS é orientado ao evento, o que quer

dizer que o estado da rede não é reportado constantemente, mas sim somente quando ocorre uma mudança de estado, o que reduz drasticamente o volume de tráfego na rede. Define perfis para a ligação de dispositivos de I/O a PLC`s numa perspectiva de comunicação ponto a ponto entre um "master" e um nó remoto de I/O.

Em seguida apresentam-se várias tabelas onde se efectua uma comparação entre as várias camadas de aplicação definidas atrás:

Nome	Ano / Introdução	Standard
DeviceNet	Março de 1994	ISO 11898 e 11519
SDS	Janeiro de 1994	Honeywell specification submitted to IEC, ISO 11989
CanOpen	1995	CiA
CanKingdom	1991	ISO 11898

Tabela 1 - Informação Geral

Nome	Topologia	Meio Físico	Máx. Nós	Máx. Distância
DeviceNet	Linha Ramificada	Par entrelaçado para sinal e potência	64 nós	500 m (b. r. variável) 6 Km com repetidores
SDS	Linha	Par entrelaçado para sinal e potência	64 nós; 126 endereços	500 m (b. r. variável)
CanOpen	Linha	Par entrelaçado + opcional, sinal e potência	127 nós	25 – 1000 m (b. r. variável)
CanKingdom	Linha	Par entrelaçado	255 nós	1000 m (b. r. variável)

Tabela 2 - Características Físicas

Nome	Método Comunicação	Transmissão	Tamanho de dados	Detecção de erros
DeviceNet	Master/Slave, multi-master, ponto a ponto	500, 250, 125 Kbps	8 bytes variável	CRC check
SDS	Master/Slave, ponto a ponto, multi-cast, multi-master	1M, 500, 250, 125 Kbps	8 bytes variável	CRC Check
CanOpen	Master/Slave, ponto a ponto, multi-cast, multi-master	1M.... 10Kbps	8 bytes variável	15 bit CRC
CanKingdom	Master/Slave	Qualquer, 125 Kbps	8 bytes variável	CRC check

Tabela 3 - Mecanismo de Transporte

7. Escolha do hardware

7.1 Escolha do Micro

Depois de escolhida a tecnologia (protocolo) que iríamos utilizar na comunicação do "nosso" nó (módulo de instrumentação) com o módulo master (PC + placa CAN), passamos ao passo seguinte, que foi a pesquisa no mercado de todas as soluções possíveis.

O primeiro passo foi a escolha de um microcontrolador que controlasse o nó CAN desenvolvido por nós.

Assim, após muita investigação, e de termos uma perspectiva da variedade de componentes (microcontroladores) actualmente comercializados, das suas características e custos, elaboramos uma tabela que será apresentada de seguida, onde colocamos todos os micros que

poderiam ser uma possível escolha. Esta tabela encontra-se dividida em fabricantes, modelo, características gerais, encapsulamento, aspectos positivos e aspectos negativos.

Nesta tabela já não fazemos referência a micros sem CAN e a controladores com CAN isolados, apesar de num primeiro passo a nossa atenção ter recaído sobre este tipo de solução, visto ser a mais antiga e com maior número de documentos de ajuda. Assim, esta tabela teve como principal critério a integração no mesmo chip do micro e do controlador CAN, pois desta forma é mais fácil a gestão de toda a rede, o que se traduz num melhor desempenho e num menor tempo de desenvolvimento.

Um segundo critério que nos ajudou na escolha da solução final foi a família a que pertence. No nosso caso optamos pela família 8XC51, evitando assim micros baseados na família C16/ST10. Esta escolha é justificada pela anterior experiência em microprocessadores da família 8XC51.

O terceiro critério que ponderou a nossa escolha foi a facilidade de aquisição em tempo útil e o encapsulamento, neste ponto eliminamos de imediato a possibilidade de micros da Philips devido à enorme dificuldade de aquisição no mercado português.

Fabricante	Modelo	ROM	RAM	I/O serie	CAN	Freq. Clock
Dallas	DS80C390	---	4K bytes	Síncrono e assíncrono	2.0B	40Mhz
Intel	87C196CA	32K bytes OTP	256 bytes	UART	2.0B	18Mhz
	87C196CB	56K bytes	256 bytes	UART	2.0B	20Mhz
Infineon	C505C	16K bytes	245 bytes	USART e porta série síncrona	2.0B	20Mhz
	C515C	64K bytes	256 bytes	Interface série síncrono, compatível com SPI	2.0B	10Mhz
	C164CI	64K bytes OTP	2K bytes	Síncrono e assíncrono	2.0B	20Mhz
	C167R	---	2K bytes	Síncrono e assíncrono	2.0B	33Mhz

Fabricante	Modelo	ROM	RAM	I/O serie	CAN	Freq. Clock
Infineon	C164CI	64K bytes OTP	2K bytes	Síncrono e assíncrono	2.0B	20Mhz
	C167R	---	2K bytes	Síncrono e assíncrono	2.0B	33Mhz
Philips	XAC37	32K bytes	32k bytes	UART, SPI	2.0B	32Mhz
	8xC591	16K bytes	512 bytes	UART, I2C	2.0B	16Mhz
	8xC592	16K bytes	512 bytes	UART	2.0B	16Mhz
	8xC598	32K bytes	512 bytes	UART	2.0B	16Mhz

Tabela 4 - Comparação dos micros com CAN

Um quarto e último critério, foi a velocidade que o micro poderia atingir e o facto de não possuir A/D interno, já que vamos usar um A/D externo de 24 bits (AD7731 da Analog Devices).

Com todos estes critérios em mente e depois de alguma discussão optamos finalmente e após algum tempo, pelo micro DS80C390 da Dallas Semiconductors.

As principais características deste micro são: 2 controladores full-function CAN 2.0B incorporados; 80C52 compatível; arquitectura de alta velocidade ou seja com possibilidade de velocidades do cristal oscilador até 40Mhz, o que na realidade equivale a velocidades de processamento de 100 ns; 2 portas série; 3 timers / counters; 16 interrupções com 3 níveis; 32 linhas I/O mais barramento de endereços / dados; 256 bytes on-chip RAM mais 4 KB on-chip SRAM; possibilidade de endereçamento acima de 4 MB de memória externa, 68 pinos PLCC e sem A/D interno.

O que tomou a nossa maior atenção foi a velocidade de processamento que atingia, aproximadamente 2,5 vezes mais do que a velocidade de um 80C51 normal.

A principal desvantagem apontada por nós neste micro, é a falta de memória interna (EPROM), assim o uso de memória externa foi inevitável.

7.2 Memória EPROM

Este é um componente fundamental no projecto visto que o micro não possuir memória de programa interna. A grande preocupação desde o início foi ter uma memória que fosse suficientemente rápida em termos de tempos de acesso, de modo que fosse totalmente compatível com o micro (DS80C390) que estamos a utilizar.

Após uma consulta ao mercado e outra consulta através de um e-mail enviado ao suporte técnico da Dallas, concluímos que a memória teria de ser o mais rápida possível. A memória aconselhada pelo suporte técnico da Dallas recaia para uma memória de alta velocidade.

Com estes dados consultamos a base de material existente no laboratório, e decidimos utilizar uma memória de 32 Kb da AMD, a Am27C256, que tem velocidades de acesso inferiores a 45 ns.

A memória é um dos componentes principais e fundamentais no sistema, pois é ela que contém todo o código do programa que vai configurar o nó de forma a poder comunicar correctamente.

7.3 Memória RAM

A memória RAM é um dos outros componentes fundamentais do projecto, pois é aqui que irá ficar armazenada toda a informação dos dados recolhida pelo A/D, e que em seguida irá ser enviada via CAN para o concentrador de forma a poder ser visualizada e interpretada.

O principal critério na escolha da RAM foi a velocidade de acesso e a capacidade.

Após consulta de mercado e a ajuda do suporte técnico da Dallas, optamos pela DS1230 da Dallas, que é uma NonVolatile SRAM que tem tempos de leitura e de escrita compatíveis com o DS80C390.

A primeira escolha foi a K6R1008C1C-VC10 da Samsung, mas depressa abandonamos esta opção devido à dificuldade existente na aquisição deste componente.

7.4 LATCH

A latch é um componente que apenas foi utilizado devido a uma opção de projecto, pois existia a possibilidade no micro de a parte mais baixa dos endereços não estar multiplexada com o barramento de dados como é usual no 80C51. Por opção colocamos na porta 0 os dados e a parte menos significativa dos endereços, mas poderíamos ter na porta 0 somente os dados, e a parte menos significativa dos endereços passaria para a porta 1. Desta forma o esquema seria mais simples mas não teríamos a porta 1 para uso genérico.

A porta 1 do microprocessador vai ser utilizada para controlar o Ad7731.

A latch que utilizamos foi a 74F573, é uma latch que foi escolhida pelo nosso conhecimento acerca da sua utilização.

7.5 Selecção do A/D

Para a selecção do conversor anológico-digital tivemos em consideração a necessidade de mais do que 16 bits de resolução na conversão, assim como uma frequência de amostragem superior a 1Khz, e outros pormenores como precisão, n.º de entradas, ganho programável, etc. Dado isto efectuamos uma pesquisa de selecção resultando daí alguns candidatos que se mostram em seguida:

Modelo	Resolução (bits)	INL (%)	N.º de entradas Dif/Ps	Freq. de amostragem máxima	Tensão de alimentação (V)	Ganho prog.
MAX1400	18	0.0015	3/5	4.8Khz	+5	Sm
MAX1401	18	0.0015	3/5	4.8Khz	+3	Sim
MAX1402	18	0.0015	3/5	4.8Khz	+5	Sim
MAX1403	18	0.0015	3/5	4.8Khz	+3	Sim
AD7714	24	0.0015	3/5	1Khz	[+3; +5]	Sim
AD7730	24	0.0015	1/2	1.2Khz	+5	Sim
AD7731	24	0.0015	3/5	6.4Khz	+5	Sim
ADS1216	24	0.0015	4/7	0.78Khz	[+3; +5]	Sim
ADS1210	24	0.0015	4/7	15.6Khz	+5	Sim

Tabela 5 - Comparação dos conversores A/D

Os potenciais candidatos finais seriam os seleccionados na tabela anterior, e a escolha recaiu sobre o ADS1210 já que proporcionava a maior frequência de amostragem e maior numero de entradas, mas uma vez que nos nossos laboratórios já possuíamos o AD7731 utilizamos este último já que a segunda escolha recaia sobre o mesmo.

7.6 Transceiver CAN

O transceiver para o barramento CAN é muito importante uma vez que sem ele os níveis de tensão do barramento e do nó entrariam em conflito, o que poderia causar grandes problemas. O transceiver adapta os sinais do controlador CAN que normalmente utiliza níveis TTL aos níveis usados pelo nível físico.

Após alguma pesquisa no mercado optamos pelo Philips 82C250, que é um transceiver muito popular, implementa o nível físico ISO 11898 e pode ser usado com velocidades acima de 1 Mb/s.

Em seguida apresentamos uma tabela com alguns dos transceivers existentes no mercado.

Fabricante	Bosch	Philips Semiconductors		SGS-Thomson	Texas Instruments	
Modelo	CF150B	82C250	82C251	L9615	SN65HVD230	SN65HVD232
Data rate max [Mbd]	0.5	1	1	0.5	1	1
ESD[kV]	2	2	2.5	2	4	4
delay [ns]	230	170	170	230	70	70
Fan out	32	64	64	32	120	120
packaging	SOIC-8	SOP-16	SO-8, DIP-8	SO-8	SO-8	SO-8

Tabela 6 - Comparação entre vários transceivers

8. Escolha do Sistema de Alimentação

Na escolha do sistema de alimentação e após algumas conversas com o orientador Professor Adriano Carvalho, ficou definido que devido ao nosso sistema ser remoto, a melhor forma de proporcionar a alimentação

seria através de baterias alimentadas, ou seja, baterias recarregáveis a partir de um painel solar.

Na primeira fase escolhemos qual a bateria que deveríamos utilizar.

8.1 Escolha da Bateria

Existem diversos tipos de baterias, mas aqui somente fazemos referência a 3 tipos, talvez por serem os mais importantes e representativos no mercado.

- Baterias de Ácido-Chumbo – são as baterias mais comuns nos automóveis, os eléctrodos são de chumbo e contêm óxidos de chumbo que se alteram durante a carga e descarga. O electrólito é uma solução diluída de ácido sulfúrico. São as baterias mais usadas em sistemas de energia solar (99% dos casos), e as que duram mais tempo (cerca de 20 anos).

- Baterias NiCa (Níquel Cádmio) – são baterias alcalinas em que o eléctrodo positivo é um óxido de Níquel e o eléctrodo negativo contém Cádmio. Estas baterias têm algumas desvantagens tais como: são bastante caras, têm perigo de derramamento de Cádmio (extremamente perigoso para o meio ambiente) e baixa eficiência. São boas escolhas se forem usadas em ocasiões espaçadas, mas a sua durabilidade diminui se a utilização for cíclica.

- Baterias NiFe (Níquel Ferro) – baterias do tipo alcalino, têm um electrólito de hidróxido de potássio, o ânodo é de malha de Chumbo com uma camada de Ferro e o cátodo é de Níquel revestido de Chumbo com um material activo de Níquel. Têm algumas desvantagens tais como: uma baixa eficiência (cerca de 50%), uma taxa muito alta de auto-descarga, um grande consumo de água e um grande volume. Nestas baterias a tensão de saída varia com a carga, muito mais do que nos outros tipos.

Existem outros tipos de baterias mas devido ao elevado preço e às suas aplicações especificas não serão aqui referidas.

Numa primeira abordagem pensamos que as baterias de NiCa fossem as melhores, talvez por serem as mais leves, ou seja, mais portáveis. No entanto têm alguns inconvenientes pois a sua eficiência não é muito elevada (65-80%) e para além deste factor, tem a desvantagem de não serem muito duradouras com cargas cíclicas devido ao efeito de memória. Assim talvez um outro tipo de bateria seria o mais apropriado.

8.2 Escolha do painel solar

Devido à aplicação ser remota pensamos que a melhor forma de dar total autonomia ao nó seria recarregar as baterias através de painéis solares.

A grande desvantagem da energia solar é o elevado custo, por tal facto só a adoptamos devido à localização remota da instrumentação sem a possibilidade de acesso à rede eléctrica.

Após alguma pesquisa pelo mercado escolhemos o modelo SM10 da Siemens Solar que usa a tecnologia cristalina (“Crystalline Silicon”). A outra tecnologia em que podia ser implementada era “Thin Film” que tem a desvantagem de ser menos eficiente e assim utilizar mais área.

O SM10 tem uma potência máxima de 10W, corrente de saída de 0,61 A, temperatura de operação de –40 a +85ºC e um peso de 1,8 Kg. Este último talvez terá sido o parâmetro que nos levou a escolher este painel solar em detrimento de outros.

Com este sistema Painel Solar + Bateria temos um sistema de alimentação remota. Na placa de controlo existe um regulador de tensão que permite a entrada de 9 a 18 V variável, e que nos dá uma tensão de saída constante de 5V, que irá servir para alimentar toda a lógica do nó.

9. Diagrama funcional

Após a selecção dos diversos componentes a integrar o nó inteligente, foi necessário definir a configuração do mesmo por forma a torná-lo uma realidade física.

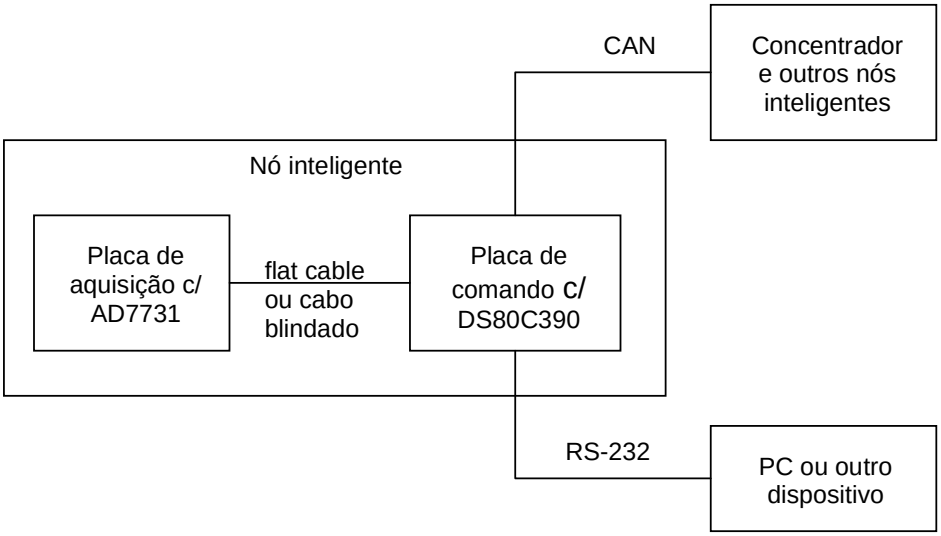

Fig. 15 - Diagrama de blocos geral

Foram concebidas duas placas, a placa principal denominada placa de comando e a placa de aquisição. A separação do conversor analógico-digital do dispositivo inteligente (microprocessador) prendeu-se com o facto de possibilitar a colocação do dispositivo de aquisição num local menos espaçoso e ainda com a maior facilidade de se tornar imune ao ruído a aquisição.

Como o próprio nome sugere a placa onde esta centralizada toda a gestão do nó é a placa de comando, sendo os seus elementos principais os seguintes:

1. Microprocessador de alta velocidade (40Mhz de frequência de relógio e 100ns de ciclo máquina) incorporando dois portos CAN independes;

2. Memória:
    a. De dados e programa com 128K bytes (NonVolatile SRAM de 70ns),
    b. De programa com 32K bytes (Eprom de 45 ns);
3. Interface de comunicação série recorrendo ao MAX232;
4. Transceivers 82C250 para o interface de comunicação CAN;

Quanto ao interface com a placa de aquisição, este é feito por flat cable, no qual seguem os sinais (DIN, DOUT, RDY, RESET...) necessários ao controlo do chip de aquisição. De notar que se a aplicação o exigir poderemos substituir o flat cable por um cabo blindado.

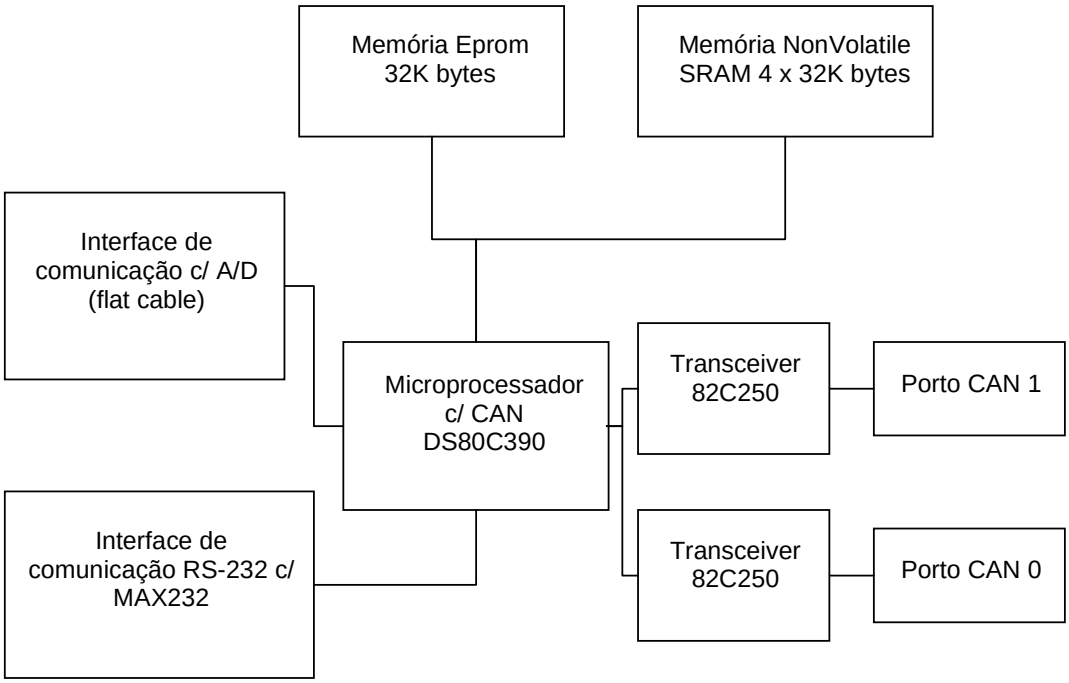

Fig. 16 - Diagrama de blocos da placa de comando

Abordando agora a placa de aquisição, o seu elemento central é o AD7731, o qual é responsável pela amostragem das grandezas que se pretendem adquirir. Toda a comunicação entre esta placa e o DS80C390 é efectuada por meio de buffers, prendendo-se a necessidade de utilização

destes com o facto do microprocessador estar numa placa distinta e possivelmente um pouco afastado do conversor.

10. Esquemáticos

10.1 Placa de comando

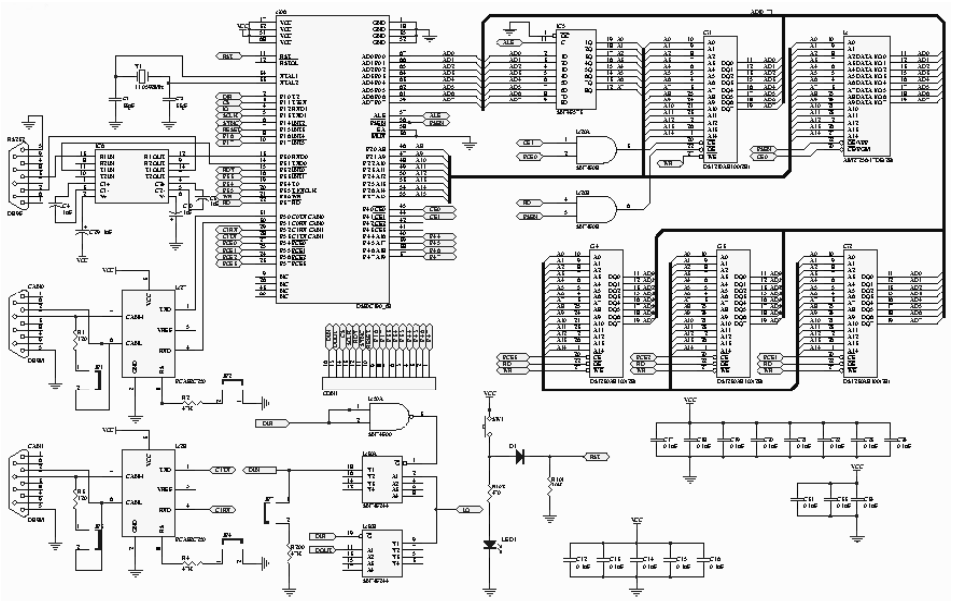

Fig. 17 - Esquema geral da placa de comando

No esquema anterior a fonte de alimentação não se encontra visível, uma vez que o documento esquemático da fonte de alimentação é anexo ao esquema principal.

10.2 Fonte de alimentação

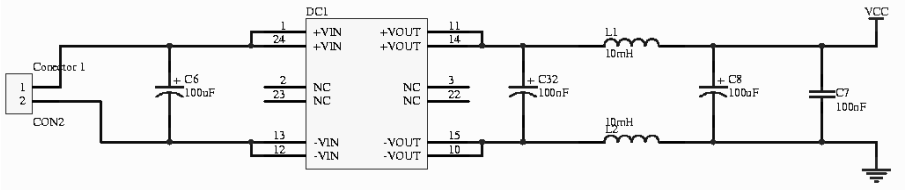

Fig. 18 - Fonte de alimentação

Uma vez que o nó pode receber uma tensão de alimentação não regulada (entre 9 a 18V), escolhemos como regulador o conversor DC/DC Tel 3-1211 da Traco. Trata-se de um conversor que oferece na sua saída uma tensão de 5V estabilizada e uma corrente máxima de 600mA, sendo esta corrente um dos factores que nos levaram à escolha deste conversor. Quanto ao filtro colocado após o conversor, este foi colocado com o intuito de estabilizar e eliminar possíveis ruídos na tensão de alimentação.

10.3 Circuito de reset

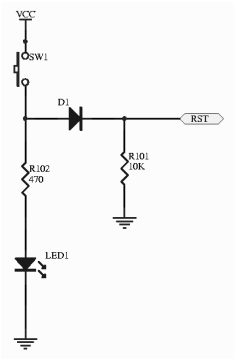

Fig. 19 - Circuito de reset

Para implementação do circuito de reset do microprocessador tomamos em consideração o facto do mesmo ser activo a nível lógico alto, assim como não efectuar um reset no arranque do sistema, pois numa pequena falha energética o nó inteligente seria reinicializado, o que em algumas situações não é o desejado.

10.4 Comunicação série RS-232

A comunicação série via RS-232 foi implementada no porto 3 do microprocessador, uma vez que o porto 1 foi utilizado para a comunicação série entre conversor A/D e o DS80C390. No que diz respeito ao adaptador de níveis, utilizou-se o vulgar MAX232 por forma a compatibilizar os níveis de tensão do microprocessador com os da porta série.

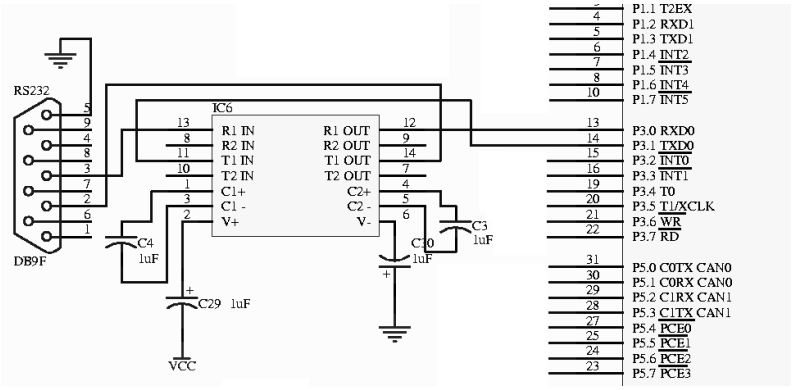

Fig. 20 - Circuito de comunicação série RS-232

10.5 Comunicação CAN

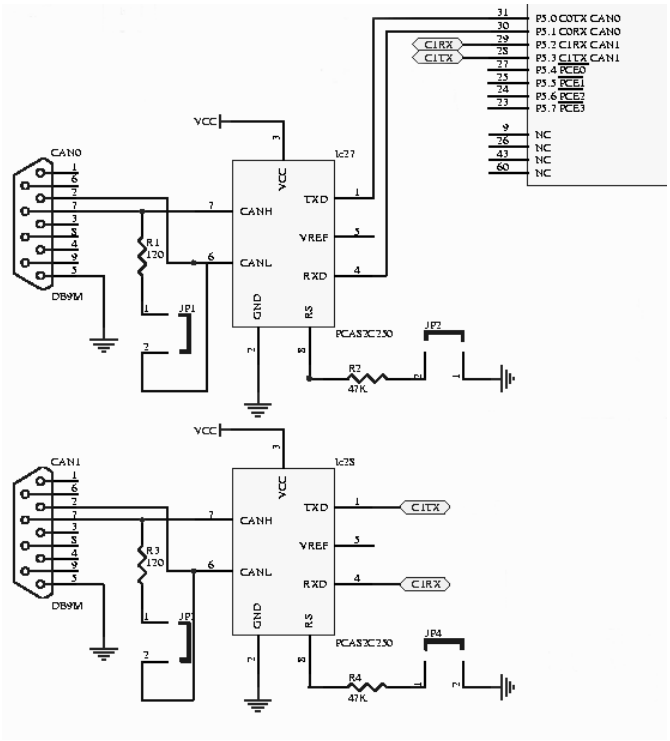

Fig. 21 - Circuito dos transceivers

No circuito de comunicação CAN utilizamos os transceivers 82C250 pelas razões já descritas na selecção de hardware. Como podemos ver temos dois portos CAN que tanto internamente (dentro do microprocessador) como externamente (fora do microprocessador) são totalmente

independentes. De referir ainda que os dois portos CAN do microprocessador encontram-se nos 4 bits menos significativos do porto 5.

10.6 Circuito para boot loader e ligação das memórias

Por forma a flexibilizar a programação do nó de instrumentação, implementamos uma solução de boot loader, a qual após o arranque do sistema permite carregar e/ou actualizar o programa residente no nó através do interface RS-232. Esta solução necessita de uma ligeira alteração nas ligações do primeiro modulo de memória ao microprocessador em relação às ligações convencionais, a qual passa por implementar um circuito lógico que permita o acesso à primeira RAM como se fosse uma ROM, ou seja, para leitura de instruções de programa.

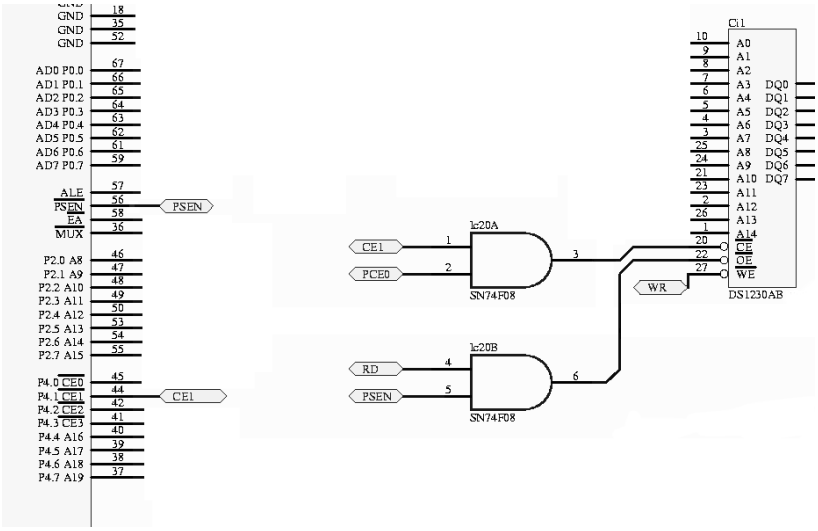

Fig. 22 - Circuito de boot loader

No que diz respeito à ligação das memórias aos respectivos sinais de controlo, o DS80C390 disponibiliza sinais de selecção e controlo para os módulos de memória já descodificados, o que evita a habitual lógica de descodificação para as memórias.

Para a memória de programa os sinais são o CE0, CE1, CE2 e CE3. Para a memória de dados são o PCE0, PCE1, PCE3 e PCE4, permitindo em

ambos os casos endereçar módulos até 1 Mbyte de capacidade. No que diz respeito ao nó, utilizamos todos os sinais PCEx e apenas os dois primeiros CEx uma vez que utilizamos 4 módulos de SRAM com 32 Kbytes cada e uma Eprom de 32 Kbytes.

Não se fará referência à implementação das ligações da latch uma vez que são as típicas, assim como no acréscimo de um condensador de acoplamento em cada chip de memória, na latch e na maior parte dos restantes circuitos integrados.

10.7 Circuito de selecção do A/D

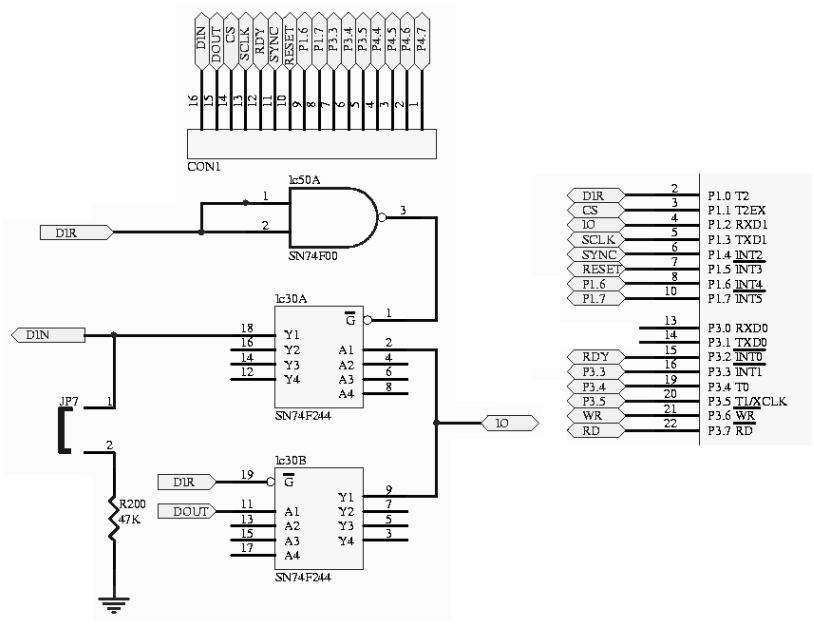

Fig. 23 - Circuito de selecção do sentido de comunicação entre A/D e Microprocessador

Para a recepção de instruções e saída de dados o AD7731 disponibiliza uma linha de entrada e outra distinta de saída para além da linha do clock. Dado que no DS80C390 a comunicação série é a típica, ou seja, a linha de entrada de dados é a mesma que de saída, surgiu a necessidade de contornar este problema. De referir que o A/D permite ligar as duas linhas de entrada e saída juntas, mas isto não permitiria um funcionamento contínuo.

Dado isto e por forma a garantir uma maior robustez na comunicação entre estes dois dispositivos introduzimos uma lógica de selecção utilizando uma porta lógica not e dois buffers tri-state. Assim controlamos o sentido da comunicação em cada instante através do bit 0 do porto 1. Na figura anterior vemos ainda os sinais que são disponibilizados no conector 1 para a ligação à placa de aquisição.

10.8 Placa de aquisição

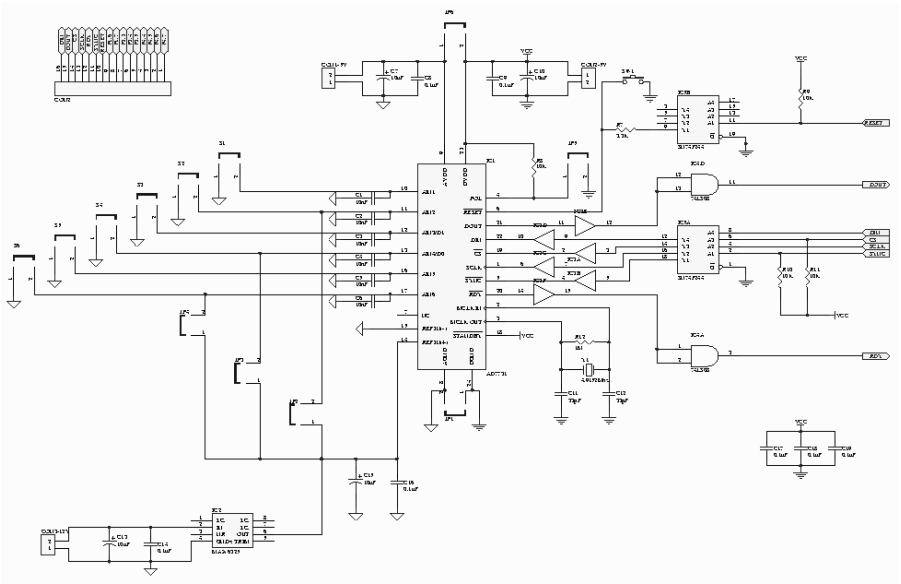

Fig 24 - Placa de aquisição de sinal

A placa de aquisição compreende o conversor AD7731, o qual possui 6 entradas, podendo estas ser configuradas como 3 entradas diferenciais ou então 5 entradas pseudo-diferenciais.

A tensão de referência é dada pelo MAX6325 caso seja possível ter uma alimentação de 12 V, caso a alimentação seja distinta deste valor utiliza-se um AD780 cuja alimentação é de 5V.

Um aspecto a relembrar e que justifica os jumpers 2, 3 e 4 é que quando se pretende adquirir tensões diferenciais é necessário ligar o terminal negativo do sinal de aquisição a 2,5V.

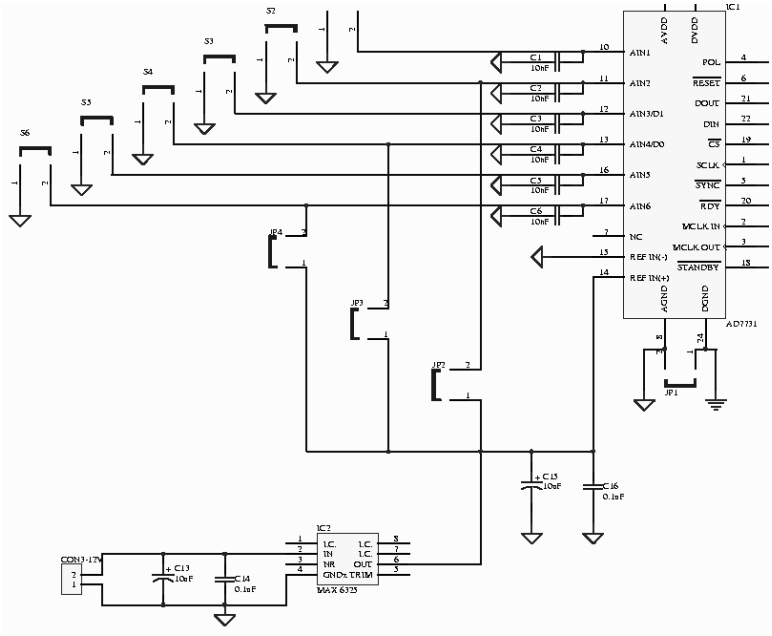

Fig. 25 - Circuito do IC da tensão de referência

De referir que este circuito possui três alimentações separadas. Uma de 12V e/ou 5V para o IC que fornece a tensão de referência, outra de 5V para o circuito digital e outra também de 5V para alimentar o circuito analógico. No limite poderá ser ligada apenas uma tensão de 5V comum para todas as partes. De referir ainda que esta separação entre alimentação analógica e digital prende-se com o facto de proteger os sinais analógicos de ruído.

A utilização dos buffers nos sinais de controlo justifica-se pelo facto do microprocessador estar numa placa distinta e possivelmente afastado do conversor, tendo com isto maior flexibilidade e imunidade ao ruído nas comunicações.

10.9 Selecção da frequência do Clock

Quanto à selecção da frequência do clock para o microprocessador, esta foi um pouco problemática uma vez que era necessário compatibilizar o timing da porta série RS-232 que comunica com o PC com o timing do CAN. Inicialmente optamos por um clock de 11.0592MHz com o qual a

configuração da porta série ficava optimizada (tolerância temporal da comunicação inferior aos 10% permitidos neste tipo de comunicação), mas o timing do CAN era problemático e impossível de compatibilizar com a porta série. Após isto e depois de alguns cálculos mais elaborados chegamos à conclusão que um clock de 18MHz compatibilizaria estes dois periféricos, mantendo inferior a 10% o erro do timing da porta série e permitindo configurar o CAN para qualquer baud rate padrão.

11. Interligação do Sistema ao Concentrador

Tendo sido definido à partida que o concentrador seria um PC, dado as vantagens inerentes, restava estabelecer o modo como a rede de CAN seria interligada com o PC.

Numa primeira abordagem, e seguindo um caminho à partida mais lógico, pensamos em utilizar uma placa de Interface CAN-ISA que se encontrava no Laboratório de Projectos de APEL. No entanto, após uma breve conversa com um elemento de um grupo de projectos que se encontrava a utilizar a referida placa, apercebemo-nos que tal utilização seria impossível, devido à impossibilidade de partilha da placa e à inexistência de drivers.

Visto que era impossível utilizar a referida placa, decidimos implementar uma interface própria, tendo surgido diversas alternativas.

A primeira alternativa ponderada seria realizar uma interface paralela num nó, a qual ligaria à porta paralela do PC. Esta implementação, que à partida poderia funcionar, não satisfazia parte dos requisitos necessários, e tratava-se de uma comunicação pouco robusta, a não ser que se utilizasse um protocolo de correcção e detecção de erros. Outra grande falha na comunicação via porta paralela, está relacionada com a velocidade de transferência de dados, aproximadamente de 1 Mbyte/s, valor que baixa consideravelmente se se utilizar detecção e correcção de erros (“over heading”).

Posta de parte toda a comunicação via porta paralela, colocamos a hipótese de utilizar uma interface série de alta velocidade tal como USB ou fire-link. Quer uma, quer outra tecnologia possibilitam a interligação de vários dispositivos no meio, existindo a possibilidade de velocidades de transferência da ordem dos 100 Mbytes/s, ou superiores no caso do Fire-link. Como alternativa válida passamos por estudar estes protocolos, e desde logo nos apercebemos das dificuldades que lhes estavam associadas. A principal desvantagem, se é que pode ser encarada como desvantagem, prende-se com o desenvolvimento de software que possibilite o controlo da mesma. Desta modo, deparamo-nos com uma extrema dificuldade em encontrar informação sobre como realizar drivers, e mesmo como realizar aplicações com suporte a esta tecnologia.

Colocadas de parte estas tecnologias, restava-nos apenas voltar para uma solução de interface via ISA ou PCI. Com esta nova solução teríamos que optar pelo barramento ISA ou PCI. Como a maior parte dos fabricantes de boards para PC praticamente abandonaram o barramento ISA, pensamos em realizar uma interface PCI. Esta tentativa revelou-se frustrada, onde para além da pesquisa sobre o barramento PCI pouco mais se fez, pois apercebemo-nos da dificuldade da utilização do barramento PCI. Trata-se de um barramento de alta velocidade, de alto desempenho com o inconveniente de muitos dos pinos se encontrarem multiplexados e o acesso ao barramento ser feito por um protocolo extremamente complicado. Existem fabricantes como é o caso da Atmel que vendem circuitos integrados específicos para estas aplicações. Mais uma vez avaliamos a possibilidade de implementar uma placa com o auxilio deste chip, no entanto abandonamos esta ideia dado que este chip apenas se encontra disponível na versão de soldadura superficial SMD.

Colocada de parte a possibilidade de realizar uma placa PCI voltamos a pensar em realizar uma placa de Interface ISA. Mesmo sendo um barramento obsoleto, o barramento ISA é extremamente versátil, os 3 barramentos que possui encontram-se dispostos em pinos separados, levando a uma simplificação de toda a lógica de interface com os dispositivos

de hardware. Uma outra vantagem deste barramento centra-se na possibilidade de ser facilmente controlado com instruções I/O, tais como “inportb(ox.....)” do C. Dadas estas facilidades tentamos projectar uma placa de adaptação ISA-CAN com o auxilio do integrado SJA-1000 da Philips. De referir que este integrado faz parte da família dos controladores CAN do tipo “stand-alone”, este integrado possibilita a comunicação compatível com CAN 2.0A ou CAN 2.0B.

O modo de operação deste integrado segue a metodologia usual para os controladores CAN. É necessário realizar uma inicialização/configuração dos registos e seguidamente escrever, ou ler, para os registos correspondentes.

No modo 2.0A, ou seja standard, o SJA-1000 tem uma zona de memória de 32 bytes que permitem controlar o dispositivo na totalidade, de referir que no modo 2.0B este espaço sobe para 128 bytes. Tal acréscimo deve-se ao facto de existir um buffer maior de recepção, um maior número de registos associados aos campos de identificação (CAN 2.0B tem 29 bits de identificação de tramas), mascaras, etc. Com esta situação teríamos de optar pela utilização do CAN 2.0A ou CAN 2.0B , uma vez que o CAN 2.0B apenas apresenta vantagens em sistemas assentes numa camada de aplicação complexa, dado ao seu maior “overheading”, optamos pelo CAN 2.0A, ou seja teríamos que possuir 32 endereços de I/O. Após uma breve análise aos portos de I/O que não se encontram atribuídos deparamo-nos com mais um problema, num computador com arquitectura PC não existem 32 endereços de I/O livres, e os que se encontram livres num determinado computador não são os que se encontram noutro. Ainda numa tentativa de viabilizar o fabrico desta placa de expansão pensamos em utilizar uma lógica de selecção de endereços, jumpers aliados à memória FIFO. Com isto, poderíamos reduzir drasticamente o número de portos necessários, contudo a complexidade deste projecto poderia não ser justificada pela importância deste componente, tal como corríamos um sério risco da placa de controlo não funcionar devido a falhas no projecto.

Abandonada a possibilidade de realizar uma interface CAN optamos por comprar uma placa de interface PCI. As especificações para esta placa passariam por: compatibilidade com CAN 2.0A e CAN 2.0B, possibilitar a comunicação a diversas velocidades, ter boa documentação técnica e suporte e se possível ter mais do que um porto de interface. A escolha recaiu assim na placa PCI CAN da National Instruments.

12. Software

Depois de escolhidos os componentes e de todo o hardware estar montado, passamos à parte de software. Nesta parte tivemos que tomar algumas decisões, sendo muitas por imposições de hardware.

As opções tomadas foram: utilização de apenas 1 porto CAN no modo standard; utilização das duas portas série, uma para comunicação série via RS-232 com o PC para efectuar o boot loader e também como possibilidade para fazer a monitorização do nó e a outra para comunicar com o A/D.

A primeira configuração que efectuamos encontra-se dentro da função init(). Muitos destes registos são protegidos temporalmente e para se poder actuar neles é necessário desabilitar a protecção temporal dos mesmos, o que é feito através de uma sequência bem definida de instruções em assemby:

```
mov ta, #aah
mov ta, #55h
```

Após estas instruções podemos efectuar as alterações que pretendemos no registo protegido em causa.

O primeiro registo em que procedemos a alterações foi o P5CNT, no qual habilitou-se o porto CAN0 e desabilitou-se o porto CAN1.

	7	6	5	4	3	2	1	0
SFR A2h	CAN1BA	CAN0BA	SP1EC	C1_I/O	C0_I/O	P5CNT.2	P5CNT.1	P5CNT.0
	RW-0	RW-0	RW-0	RW-0	RW-0	RT-0	RT-0	RT-0

Fig. 26 - Registo P5CNT

Em seguida efectuamos o reset a todos os registos SFR do CAN0 (colocar o bit CRST do C0C a 0) e habilitamos a inicialização do software dos 16bytes do CAN0 MOVX SRAM (bit SWINT a 1), ambos fazem parte do SFR C0C.

	7	6	5	4	3	2	1	0
SFR A3h	ERIE	STIE	PDE	SIESTA	CRST	AUTOB	ERCS	SWINT
	RW-0	RW-0	RW-0	RW-0	RT-1	RW-0	RW-0	RW-1

Fig. 27 - Registo C0C

Após o reset e habilitação vamos efectuar os cálculos para preencher o CAN0 bus timing. Estes cálculos encontram-se em anexo numa folha de Excel. Nestes cálculos escolhemos o clock de 18MHz por ser compatível com o bus timing do CAN. Com estes valores calculados preenchemos os registos relativos ao bus timing CAN0, esses registos são o C0BT0 e C0BT1.

MOVX Address[1]	7	6	5	4	3	2	1	0
xxxx04h	SJW1	SJW0	BPR5	BPR4	BPR3	BPR2	BPR1	BPR0

Fig. 28 - Registo CnBT0

MOVX Address[1]	7	6	5	4	3	2	1	0
xxxx05h	SMP	TSEG26	TSEG25	TSEG24	TSEG13	TSEG12	TSEG11	TSEG10

Fig. 29 - Registo CnBT0

O bit timing no CAN2.0B é baseado numa unidade chamada tempo nominal de bit. Este tempo nominal de bit está dividido em 4 períodos de tempo específicos:

- SYNC_SEG é o segmento de sincronização, tem a duração de 1TQ (Time Quantum) e é usado para sincronizar os vários nós da rede.
- PROG_SEG é utilizado para efectuar a compensação dos atrasos ocorridos ao longo da rede.

- PHASE_SEG1 e PHASE_SEG2 são utilizados para compensar os erros de fase e podem ser alterados através dos bits SJW1 e SJW0 no registo C0BT0.

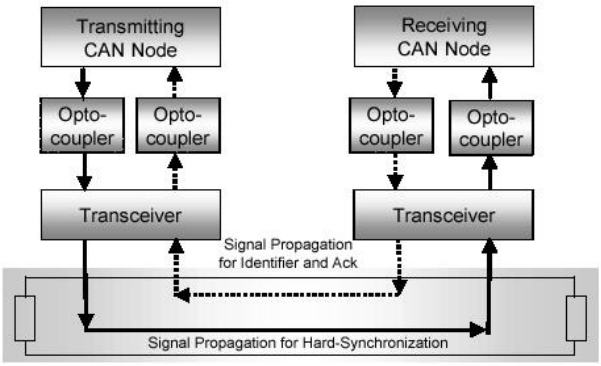

Fig. 30 – Propagação do sinal

O PROP_SEG é necessário para compensar os atrasos na propagação do sinal através dos cabos e de interfaces electrónicos dos nós (microcontroladores). A soma dos atrasos de propagação do controlador CAN, isolamento galvânico (opcional), transceivers e linhas tem que ser menor do que o segmento de tempo de propagação (PROP_SEG) dentro de um bit.

Estes atrasos são combinados duas vezes, por causa da hard synchronization. Assim vem que:

$$t_{propagação} = 2(t_{cabo} + t_{controlador} + t_{optoacoplador} + t_{transceiver})$$

O valor do bit vai ser amostrado no ponto de amostragem. O Time Quantum (TQ) é uma unidade derivada da divisão do valor do cristal do

oscilador do microcontrolador pelo Baud Rate Prescaler (programado pelos bits BPR5-BPR0 do C0BT0) e o System Clock Divider (programado pelos bits SCD2-SCD0 do C0R). Combinando o PROP_SEG e o PHASE_SEG1 num único período designado por T_{TSEG1} e igualando a SYNC_SEG a t_{SYNC_SEG} e PHASE_SEG2 a t_{SEG2} temos as bases para o cálculos dos segmentos de tempo mostrados na figura do bit timing.

O C0BT0 (CAN 0 Bus timing 0) contém os bits de controlo para o PHASE_SEG1 e PHASE_SEG2 tal como os bits para o Baud Rate Prescaler (BPR5-0). O registo C0BT1 (CAN 0 Bus timing 1) controla a frequência de amostragem. O valor de ambos os registos é carregado todas as vezes que se modifica o valor do bit SWINT de 1 para 0. Estes parâmetros têm que ser indicados antes do início das operações com CAN. Para evitar situações imprevisíveis nunca se devem preencher estes dois registos com todos os valores a zero.

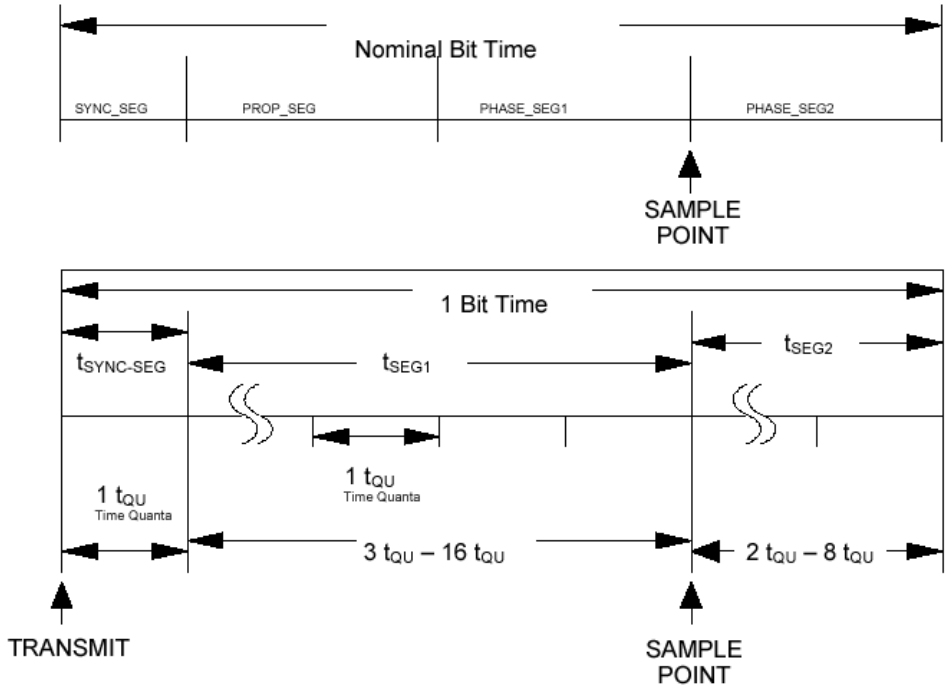

Fig. 31 - bit timing

O tempo dos vários segmentos é determinado pelas seguintes equações:

$$t_{QU} = \frac{BRPV \times CCD}{F_{OSC}}$$

$$t_{SYNC\_SEG} = 1 \cdot t_{QU}$$

$$t_{TSEG1} = (TS1\_LEN) \cdot t_{QU}$$

$$t_{TSEG2} = (TS2\_LEN) \cdot t_{QU}$$

$$t_{SJW} = (SJW) \cdot t_{QU}$$

$$t_{QU\_per\_bit} = \frac{1}{baud\_rate \cdot t_{QU}}$$

onde o BRPV é o CAN baud Rate Prescaler, F_{OSC} é a frequência de oscilação do cristal do microprocessador. Todos os outros valores foram consultados do datasheet do 80C390.

Para além destas equações temos ainda as seguintes restrições a satisfazer:

$$t_{TSEG1} \geq t_{TSEG2}$$

$$t_{TSEG2} \geq t_{SJW}$$

$$t_{SJW} < t_{TSEG1}$$

$$2 \leq TS1\_LEN \leq 16$$

$$2 \le TS2\_LEN \le 8$$

$$(TS1\_KLEN + TS2\_LEN + 1) \le 25$$

Depois de efectuados os cálculos (folha de Excel) preenchemos os registos C0BT0 e C0BT1 com os valores calculados.

Baud Rate	BRP	SJW	TSEG1	TSEG2	C0BT0	C0BT1
1 Mbit/s	1	3	5	3	80	24
500 Kbit/s	1	4	13	4	C0	3C
250 Kbit/s	2	4	13	4	C1	3C
125 Kbit/s	4	4	13	4	C3	3C
20 Kbit/s	30	4	8	6	DD	57

Tabela 7 - Tempos de bit e BTRs para 18 Mhz

Após estes cálculos colocamos de novo o bit SWINT (SFR C0C) a 0 de modo a que estes valores não possam ser alterados.

Dentro desta função definimos quais os centros de mensagens do CAN0 que iriam servir para transmitir e receber sendo respectivamente os centros de mensagens 2 e 1.

Após esta inicialização tivemos que definir como seria a nossa estrutura de mensagem para o CAN. Assim, depois de várias leituras do 80C390 userguide definimos a estrutura can_object:

```
Typedef struct
{
        unsigned char cmar0;
        unsigned char cmar1;
        unsigned char cmar2;
        unsigned char cmar3;
        unsigned char cmf;
        unsigned char packet[8];
} can_object
```

Dentro desta estrutura nos quatro primeiros campos vamos definir o valor do identificador da mensagem. No nosso caso e como estamos a utilizar o formato standard, o identificador vai ficar colocado nos bits mais significativos, ou seja, o ID 10-0 corresponde ao ID 28-18 no C0MyAR0 e C0MyAR1. Quando configurado no modo de receber estes bits vão servir de termo de comparação.

O campo cmf é utilizado para identificar, no caso da transmissão, a quantidade de bytes a transmitir no campo de dados (este n.º pode ir de 0 a 8 bytes). Podemos igualmente indicar se a trama está no formato estendido ou standard.

No último campo, o campo de dados, é onde vão estar os dados a serem transmitidos ou recebidos.

CAN 1 MESSAGE CENTER 1

	Reserved								xxxx10h - 11h
C1M1AR0	CAN 1 MESSAGE 1 ARBITRATION REGISTER 0								xxxx12h
C1M1AR1	CAN 1 MESSAGE 1 ARBITRATION REGISTER 1								xxxx13h
C1M1AR2	CAN 1 MESSAGE 1 ARBITRATION REGISTER 2								xxxx14h
C1M1AR3	CAN 1 MESSAGE 1 ARBITRATION REGISTER 3							WTOE	xxxx15h
C1M1F	DTBYC3	DTBYC2	DTBYC1	DTBYC0	T/$\overline{R}$	EX/$\overline{ST}$	MEME	MDME	xxxx16h
C1M1D0-7	CAN 1 MESSAGE 1 DATA BYTES 0 - 7								xxxx17h - 1Eh
	Reserved								xxxx1Fh

Fig. 32 - Estrutura dos centros de mensagens CAN

O passo seguinte foi a configuração da memória o que é feito na função init_mem(). Existem 32K bytes de EPROM (1 módulo) e 128K bytes de NonVolatile SRAM (4 módulos), sendo um dos módulos de SRAM para armazenar a actualização e/ou programa carregado por meio do boot loader. Posto isto, ao efectuar o programa de inicialização o primeiro passo a ser efectuado foi alterar o registo P4CNT onde se pode definir a quantidade de memória existente por cada sinal, podendo ir de 32K bytes por bloco de memória até 1 Mbyte por bloco totalizando 4 Mbytes de memória. Pretendemos ter acessível dois sinais, o CE0 e o CE1, um para a EPROM e outro para a SRAM que vai servir como memória de programa.

	7	6	5	4	3	2	1	0
SFR 92h	1	SBCAN	P4CNT.5	P4CNT.4	P4CNT.3	P4CNT.2	P4CNT.1	P4CNT.0
	R-1	RT-0	RT-1	RT-1	RT-1	RT-1	RT-1	RT-1

Fig. 33 - Registo P4CNT

O modo de endereçamento utilizado foi o modo de 16 bits com a tradicional stack pointer do 80C51, estes valores são definidos no SFR ACON.

	7	6	5	4	3	2	1	0
SFR 9Dh	1	1	1	1	1	SA	AM1	AM0
	R-1	R-1	R-1	R-1	R-1	RT-0	RT-0	RT-0

Fig. 34 - Registo ACON

Por último configuramos o registo P5CNT (bits P5CNT.2-0) por forma a existir 4 sinais de controlo para a memória de dados, cada sinal toma conta de uma memória de 32K bytes.

Quando efectuamos a transmissão de uma mensagem temos de fazer o carregamento da mensagem, para tal desactivamos o centro de mensagens, em seguida carregamos os dados para o campo de dados e enviamos as mensagens, após esperar algum tempo até que a mensagem seja enviada voltamos a desactivar o centro de envio e activamos o centro de recepção que tinha sido desactivado no início da transmissão.

Para a comunicação com o A/D utilizamos a porta série 1 no modo 0, como o modo 0 é um modo de comunicação síncrono com um tamanho de 8 bits e em que o período é de $12 \cdot t_{CLK}$ em que t_{CLK} é o período do oscilador do microcontrolador. Dado que o A/D tem um protocolo SPI (Serial Port Interface) aproveitamos a porta série 1 no modo 0 para efectuar o interface já referenciado atrás. O interface SPI é igualmente um protocolo síncrono tal como o modo 0. O seu funcionamento é essencialmente um shift register onde todos os dados que são colocados na sua entrada são passados para a saída. A programação da comunicação entre o A/D e o micro encontra-se

explicada à parte pois não faz parte da comunicação CAN (ver ponto seguinte).

Por fim na função init_can_interrupt() efectuamos a habilitação das interrupções do CAN0 (através do registo EIE dos SFRs) e das interrupções dos erros e status do CAN0 através do registo SFR C0C.

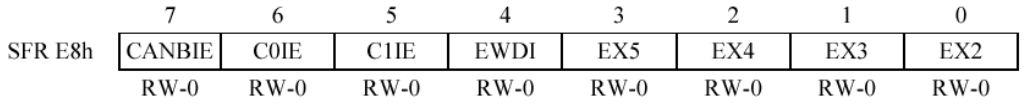

Fig. 35 - Registo EIE

Para além destes registos temos de efectuar a habilitação do bit EA do registo IE que serve para habilitar todas as interrupções. As interrupções geradas por CAN por modificação do CAN0 Status Register, C0IR, podem ser de três tipos:

- ERR_CODE onde o erro pode ser de diferentes tipos, conforme a configuração dos bits ER2-0.
- BUS_OFF em que o CAN é desabilitado totalmente e resulta do contador de erros da transmissão ultrapassar os 256 erros.
- ERR_WRN em que existe um erro para nos avisar que o contador de erros ultrapassou o limite de 96 ou 128 conforme o estado do bit ERCS do registo C0C.

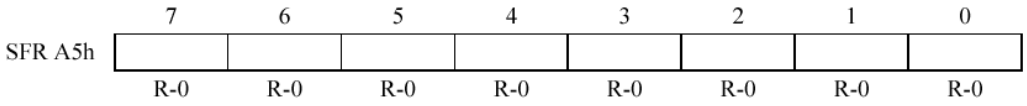

Fig. 36 - Registo C0IR

Todo o código relativo à implementação do nível físico encontra-se em anexo a este relatório. Quanto à parte de alto nível não desenvolvemos qualquer aplicação de modo que o nosso software fosse o mais genérico possível. A partir deste ponto poderia ser desenvolvido um software de

aplicação que permitisse interface com o utilizador, ou seja, que permitisse o controlo e configuração de toda a rede.

Em anexo estão também alguns programas por nós elaborados para testes. Efectuamos testes em separado, testando todo o hardware e periféricos e depois efectuamos a interligação total. Os testes em separado foram: testes de memória, testes das portas série (RS-232 e A/D), testes para o CAN (transmissão e recepção), entre outros.

Todo o código por nós gerado foi desenvolvido com recurso ao KEIL μVision2, o qual é um compilador da KEIL para a família 51.

Comunicação com o AD7731:

A função init_ad baseia a sua concepção no fluxograma de Escrita para o A/D, acrescentando apenas alguns pormenores como seja a variável dir que é responsável pelo sentido da comunicação. Apesar do AD7731 possuir duas linhas separadas para I/O o microprocessador não as possui, como tal foi necessário implementar uma lógica de selecção por forma a evitar situações indesejadas tais como a possibilidade de ler em modo continuo. Outro acréscimo que não foi especificado no fluxograma foi a variável REN1, a qual é responsável por habilitar o microprocessador a enviar pela porta série 1 no modo 0 os comandos do A/D.

Passando agora para a função read_ad, ou seja, para a função de leitura das amostras, tal como a função anterior esta baseia a sua estrutura no fluxograma de leitura, acrescentando apenas o controlo das variáveis dir e REN1 pelas razões enunciadas anteriormente. De realçar que como a amostra foi definida para 24 bits preenchemos um vector de 3 posições para cada amostra.

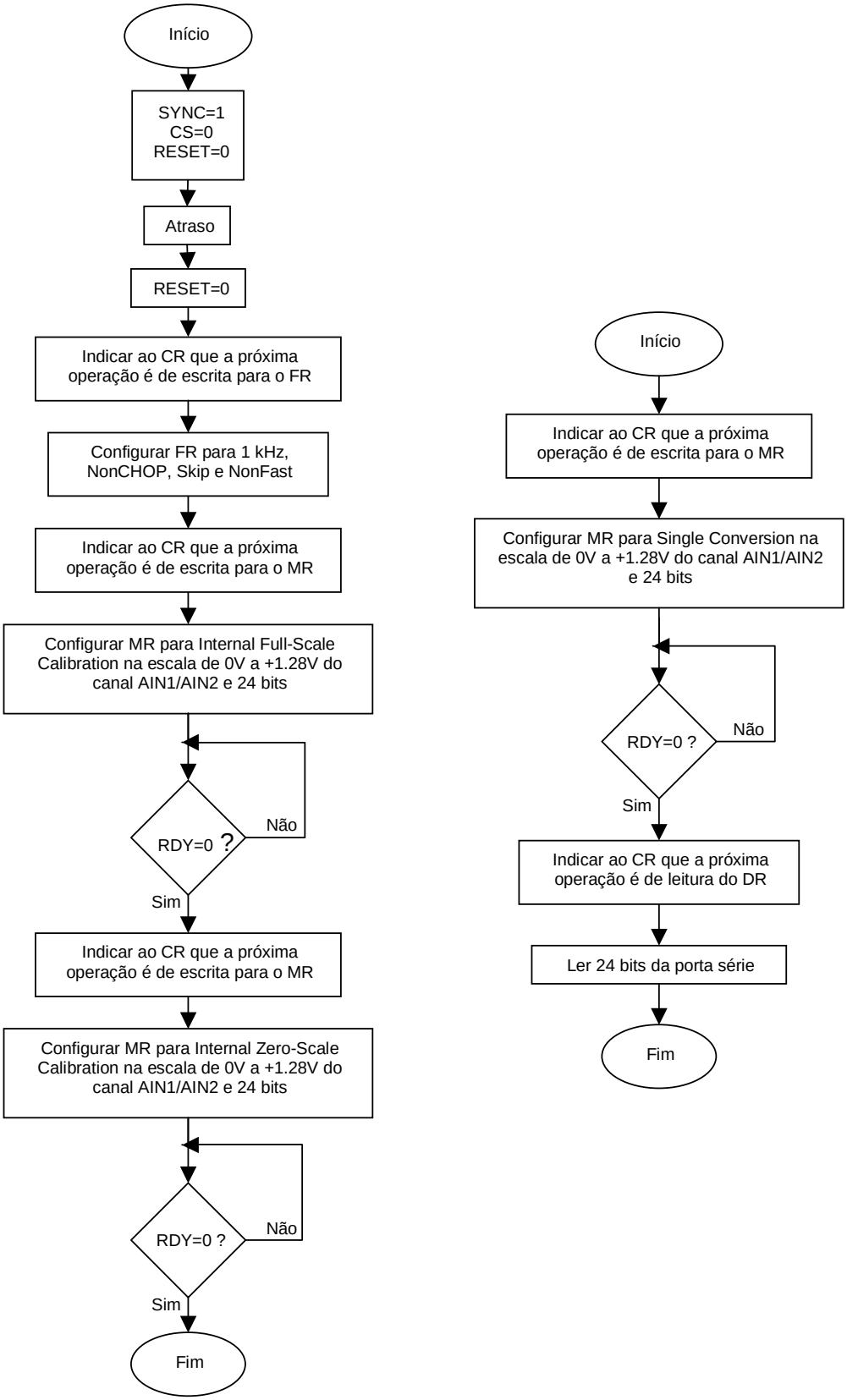

Fig. 37 - Fluxograma de Escrita e Leitura no A/D

13. Software do nó concentrador

Como foi referido o nó concentrador foi implementado com o auxílio de um PC, sendo a interface CAN realizada por intermédio de uma placa PCI.

Dada a nossa inexperiência em programação de alto nível, nomeadamente no que se refere a programação em ambiente Windows e atendendo que possuímos alguns conhecimentos de LabView, decidimos desenvolver as aplicações do nó concentrador neste software de desenvolvimento.

13.1 LabView: uma programação gráfica

A programação em LabView é extremamente eficiente e intuitiva. A visualização gráfica das primitivas de controlo de programação ajudam a compreender a envolvente de todo o programa. Trata-se de uma ferramenta muito poderosa e eficiente nomeadamente na realização de interfaces gráficas e na produção de ambientes destinados a instrumentação. O que pode ser realizado em poucos minutos no LabView pode levar horas ou mesmo dias a ser realizado noutras ferramentas de programação como o caso do Visual C++ ou outros.

A juntar a estas características acrescenta-se o facto que a National Instruments é o fabricante da placa e do LabView, pelo que asseguramos a compatibilidade de drivers.

Dado alguns contratempos no desenvolvimento do hardware esta aplicação não ficou com todas as potencialidades desejadas, apenas foi possível realizar uma aplicação capaz de testar o funcionamento remoto do nó de instrumentação.

13.2 Protocolo desenvolvido

O protocolo desenvolvido para a aplicação mantém-se no mais simples possível de modo a conseguir-se taxas de transferência elevadas sem que isso obrigue à perda de informação.

Uma vez que é possível saber se uma mensagem foi recebida correctamente (especificação CAN) não há necessidade de implementar um protocolo que garanta reenvio em caso de falha , esta capacidade é assegurada pelo CAN.

Funcionalidades de comunicação pretendidas:

- Pedido de digitalização periódico;
- Recepção de digitalizações periódicas;
- Alteração do estado das saídas digitais;
- Leitura do estado das saídas digitais;
- Reinicialização dos nós remotos.

O protocolo desenvolvido para realizar estas funcionalidades assenta numa estrutura de mestre/escravo (master/slave). Pode parecer irreal a utilização de tal modelo mas dado que as ordens de envio de amostras são dadas pelo nó concentrador assumimos uma arquitectura deste tipo.

13.3 Definição de mensagens

As mensagens definidas para realizar as funcionalidades pretendidas foram escolhidas de modo a que os identificadores realizassem o papel de arbitragem de prioridades. Neste contexto definimos os seguintes identificadores:

1560 - Ordem de Reset;

1561 - Ordem de pedido de envio de conversões;

1564 - Ordem de alteração de saídas digitais;

1566 - Ordem de leitura do estado de entradas/saídas digitais.

Como as mensagens com identificadores menores são mais prioritárias em relação a mensagens com maior identificador, é possível definir uma ordem de prioridades atendendo ao identificador atribuído a cada mensagem. Na aplicação desenvolvida a mensagem de maior prioridade é o Reset, assim o valor do identificador definido para esta mensagem foi o mais baixo. A escolha do valor dos identificadores é praticamente irrelevante pois apenas iremos ter quatro mensagens, é relevante apenas manter a ordem de grandeza dos identificadores .

Do lado do nó asseguramos que as mensagens são recebidas e que é executado o pedido, associado a cada uma delas, esta funcionalidade é assegurada por meio de interrupções. Através de uma avaliação do valor do identificador da mensagem é possível saber qual o pedido em questão (para mais detalhes consultar o anexo do programa).

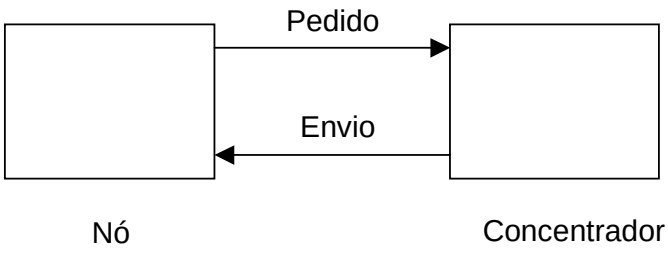

Fig. 38 – Fuxograma entre nó e concentrador

Estrutura das mensagens

As mensagens têm a seguinte estrutura:

Reset ⇒ Data Frame, com zero bytes de data

Envio Conversão ⇒ Remote Frame
O envio da conversão (nó → concentrador) é realizado com o auxilio de três bytes de data.

Leitura do estado das entradas das saídas digitais ⇒ Remote Frame

O envio do estado (nó → concentrador) é realizado com o auxilio de um byte de data.

Conversão do valor amostrado

O conversor analógico/digital AD7731 possui, tal como já foi referido, uma interface série organizada segundo o modelo “Little Endian”, ou seja, os bits mais significativos ocorrem primeiro. Como sabemos o C51 e seus derivados possuem uma tecnologia “Big Endian”, ou seja bits menos significativos primeiro.

A fim de contornar esta questão optou-se por não efectuar a inversão dos bits no microcontrolador, desta forma poupa-se algum tempo de processamento. Para conseguir converter o valor para a escala de entrada é agora necessário inverter os três bytes de dados e em seguida multiplicar cada um pelo seu peso. Esta função é implementada pelo sub-vi chamado “Converte_bases”.

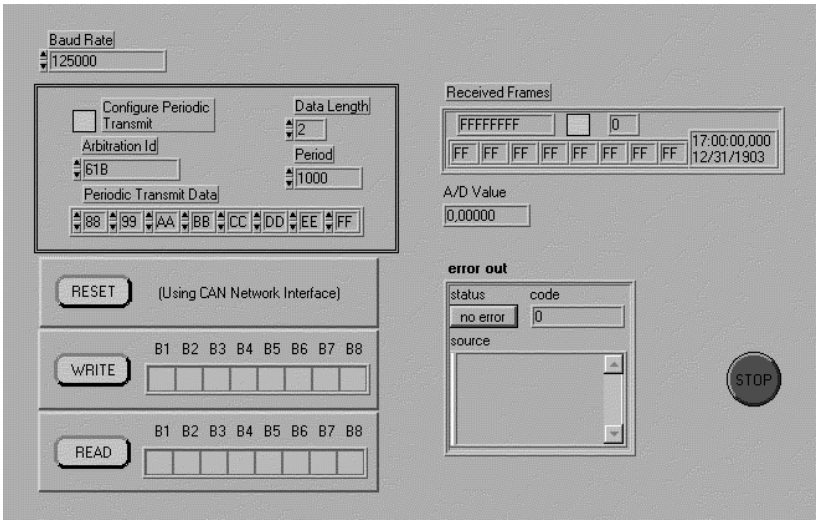

Fig. 39 - Tela de controlo local e visualização remota

14. Conclusões

A realização deste projecto permitiu-nos tomar conhecimento das dificuldades existentes na realização de um projecto de engenharia. As dificuldades surgiram desde a análise de requisitos, aquisição de componentes, planificação do projecto, desenho de esquemáticos, produção de software e hardware.

Com este projecto ganhamos uma maior sensibilidade para questões básicas como planificação e divisão de tarefas, questões estas que devem estar presentes na formação de um engenheiro mas para as quais ainda não estávamos sensibilizados. Por outro lado permitiu-nos conhecer de forma mais aprofundada o que são redes de campo e em particular o CAN.
(Controller Area Network).

Esperamos ter deixado em aberto a possibilidade de continuação do desenvolvimento deste projecto, principalmente o desenvolvimento de uma camada de aplicação e a futura interligação com bases de dados e visualização remota com controlo através da Web.

15. Bibliografia

Philips, RAM loader program for 80C51 family applications:
http://www.trimedia.philips.com/acrobat/applicationnotes/AN440.pdf

Philips, SJA1000 Stand-alone CAN controller
http://www.semiconductors.philips.com/acrobat/datasheets/SJA1000_3.pdf

Philips, PCA82C250 CAN controller interface
http://www.semiconductors.philips.com/acrobat/datasheets/PCA82C250_5.pdf

Philips, PCA82C251 CAN controller interface
http://www-us6.semiconductors.com/acrobat/datasheets/PCA82C251_3.pdf

Philips, Application note PCA82C250/251 CAN Transceiver, AN96116
http://www.trimedia.philips.com/acrobat/applicationnotes/an96116.pdf

STMicroelectronics, L9615 CAN BUS TRANSCEIVER
http://us.st.com/stonline/books/pdf/docs/5637.pdf

Texas Instruments, SN65HVD232, 3.3-V Can Transceiver
http://www-s.ti.com/sc/psheets/slos346c/slos346c.pdf

Dallas, DS80C390 Dual CAN High Speed Microprocessor
http://pdfserv.maxim-ic.com/arpdf/DS80C390.pdf

Dallas, High-Speed Microcontroller User's Guide
http://pdfserv.maxim-ic.com/arpdf/Design/hsmicro_userguide.pdf

Dallas, DS80C390 High-Speed Microcontroller User’s Guide Supplement
http://pdfserv.maxim-ic.com/arpdf/Design/80c390_userguide.pdf

Dallas, Microcontroller Design Guidelines for Reducing ALE Signal Noise
http://dbserv.maxim-ic.com/appnotes.cfm?appnote_number=551

Dallas, Design Guidelines for Microcontrollers Incorporating NVRAM
http://dbserv.maxim-ic.com/appnotes.cfm?appnote_number=552

Dallas, Using the High-Speed Microcontroller as a Bootstrap Loader
http://dbserv.maxim-ic.com/appnotes.cfm?appnote_number=577

Dallas, DS80C320 Memory Interface Timing
http://pdfserv.maxim-ic.com/arpdf/AppNotes/app57.pdf

Dallas, High-Speed Micro Memory Interface Timing
http://pdfserv.maxim-ic.com/arpdf/AppNotes/app89.pdf

Dallas, Memory Expansion with the High-Speed Microcontroller Family
http://pdfserv.maxim-ic.com/arpdf/AppNotes/app81.pdf

Dallas, Ds1230y 256K NonVolatile SRAM
http://pdfserv.maxim-ic.com/arpdf/DS1230AB-DS1230Y.pdf

Texas Instruments, TMS27C256 Read-Only Memory
http://www-s.ti.com/sc/psheets/smls256h/smls256h.pdf

Field bus – Comparação:
http://www.synergetic.com/compare.htm

WorldFIP FAQ:
http://www.worldfip.org/faq.html

Cabling FAQ from UseNet group comp.dcom.cabling:
http://web.uvic.ca/tats/cablespecs.html

Arquitectura MAP e Protocolo MMS:
http://penta.ufrgs.br/rc952/trab2/mms1.html

Bosch, informação sobre CAN
www.can.bosch.com

CIA, CAN Specification 2.0, Part A
http://www.can-cia.de/CAN20A.pdf

CIA, CAN Specification 2.0, Part B
http://www.can-cia.de/CAN20B.pdf

CIA, CAN Specification 2.0, Addendum
http://www.can-cia.de/CAN2AD.pdf

CIA, CAN Physical Layer for Industrial Applications
http://www.can-cia.de/DS102.pdf

CIA, CanOpen Cabling and Connector Pin Assignment
http://www.can-cia.de/DR303_1.pdf

CIA, informação sobre CanOpen e CAL
http://www.can-cia.de/

Kvaser, informação sobre CAN e a aplicação de alto nível CanKingdom
http://www.kvaser.com

CanKingdom, informação sobre a aplicação de alto nível CanKingdom
http://www.cankingdom.org/

ODVA, informação sobre Open DeviceNet
www.odva.org

Honeywell, informação sobre SDS
www.honeywell.sensing.com

Informação geral sobre CAN, com links para outras páginas sobre CAN
http://www.omegas.co.uk/CAN/
http://www.cs-group.de/
http://www.lawicel.com/e_links_can.htm

MAXIM, MAX6325 low-noise, precision voltage references
http://pdfserv.maxim-ic.com/arpdf/MAX6325-MAX6350.pdf

MAXIM, MAX232 line drivers/receivers
http://pdfserv.maxim-ic.com/arpdf/MAX220-MAX249.pdf

MAXIM, MAX1400 18-Bit Multichannel Sigma Delta ADC
http://pdfserv.maxim-ic.com/arpdf/MAX1400.pdf

MAXIM, MAX14001 18-Bit Multichannel Sigma Delta ADC
http://pdfserv.maxim-ic.com/arpdf/MAX1401.pdf

MAXIM, MAX14002 18-Bit Multichannel Sigma Delta ADC
http://pdfserv.maxim-ic.com/arpdf/MAX1402.pdf

MAXIM, MAX14003 18-Bit Multichannel Sigma Delta ADC
http://pdfserv.maxim-ic.com/arpdf/MAX1403.pdf

Analog Devices, AD7714 Signal Conditioning ADC
http://www.analog.com/pdf/AD7714_c.pdf

Analog Devices, AD7730 Bridge Transducer ADC
http://www.analog.com/pdf/AD7730_L.pdf

Analog Devices, AD7731 Low Noise, High Throughput 24-Bit Sigma-Delta ADC
http://www.analog.com/pdf/AD7730_L.pdf

Analog Devices, AD780 Ultrahigh precision band gap reference voltage
http://www.analog.com/pdf/AD780_b.pdf

Burr-Brown, ADS1216 24-Bit ADC
http://www-s.ti.com/sc/psheets/sbas171b/sbas171b.pdf

Burr-Brown, ADS1210 24-Bit ADC
http://www-s.ti.com/sc/psheets/sbas034/sbas034.pdf

Traco Power, TEL 3-1211 3W DC/DC converter
http://www.tracopower.com/products/tel3.pdf

National Instruments, informação sobre a placa PCI-CAN e LabVIEW
http://www.ni.com/

USB, informação geral
http://www.usb.org/

8052, informação sobre 80C52
http://www.8052.com

LPT, informação sobre portas paralelas
http://www.lpt.com/

Protel, software de desenho para esquemáticos e pcb
http://www.protel.com/

Eagle, software de desenho para esquemáticos e pcb
http://www.cadsoft.de/

Compiladores para o código do microcontrolador
http://www.keil.com/
http://www.tasking.com/
http://www.amrai.com
http://www.fsinc.com/
http://www.metaice.com
http://www.vaultbbs.com/pinnacle/

Assembladores
http://www.halcyon.com/squakvly/

Bibliografia tradicional

Barbosa, Manuel ;Conformance Testing Issues with Aplication to the CanOpen Protocol, University of Newcastle

Schultz, Thomas ;C and the 8051, Vol1, 2ª Ed, Prentice Hall 1998

Portugal, Paulo; Desenvolvimento de um nó inteligente de I/O analógicas e digitais para uma rede Ethernet, com suporte no protocolo TCP/IP

Tom Shanley, Don Anderson; ISA system architecture, Addison-Wesley Publishing Company 1995

16. ANEXOS

Software de teste dos portos CAN

```
dpx             equ     (093h)
;*************** CAN sfrs ***************************
c0rms0          equ     (096h)
c0rms1          equ     (097h)
c0tma0          equ     (09eh)
c0tma1          equ     (09fh)
p5              equ     (0a1h)
p5cnt           equ     (0a2h)
c0c             equ     (0a3h)
c0s             equ     (0a4h)
c0ir            equ     (0a5h)
c0te            equ     (0a6h)
c0re            equ     (0a7h)
c0m1c           equ     (0abh)
c0m2c           equ     (0ach)
c0m3c           equ     (0adh)
c0m4c           equ     (0aeh)
c0m5c           equ     (0afh)
c0m6c           equ     (0b3h)
c0m7c           equ     (0b4h)
c0m8c           equ     (0b5h)
c0m9c           equ     (0b6h)
c0m10c          equ     (0b7h)
c0m11c          equ     (0bbh)
c0m12c          equ     (0bch)
c0m13c          equ     (0bdh)
c0m14c          equ     (0beh)
c0m15c          equ     (0bfh)
ta              equ     (0c7h)
c1c             equ     (0e3h)
c1s             equ     (0e4h)
c1ir            equ     (0e5h)
c1te            equ     (0e6h)
c1re            equ     (0e7h)
c1m1c           equ     (0ebh)
c1m2c           equ     (0ech)
;*************** config registers ********************
C0MID0          equ     (0EE00h)
C0MID1          equ     (0EE02h)
C0MA0           equ     (0EE01h)
C0MA1           equ     (0EE03h)
C0BT0           equ     (0EE04h)
C0BT1           equ     (0EE05h)
C1BT0           equ     (0EF04h)
C1BT1           equ     (0EF05h)
C0SGM0          equ     (0EE06h)
C0SGM1          equ     (0EE07h)
C0EGM0          equ     (0EE08h)
C1SGM0          equ     (0EF06h)
C1SGM1          equ     (0EF07h)
C0M1AR0         equ     (0EE12h)
C0M1AR1         equ     (0EE13h)
C0M1AR2         equ     (0EE14h)
C0M1F           equ     (0EE16h)
C0M1D0          equ     (0EE17h)
C1M1AR0 equ  (0EF12h)
```

```
C1M1AR1     equ     (0EF13h)
C1M1F       equ     (0EF16h)
C1M1D0      equ     (0EF17h)

org 0h
        ljmp    start
;-------------------------------------------------
; main routine
;
;-------------------------------------------------
org 0100h
start:
        mov     sp, #40h        ;mov stack start
        mov     dpx, #00h       ;CAN MOVX = 00EE, 00EF
        mov     ta, #0aah
        mov     ta, #55h
        orl     p5cnt, #18h     ;enable CAN0 and CAN1
;-------------------------------------------------
;CAN1 configuration
;-------------------------------------------------
        mov     ta, #0aah
        mov     ta, #55h
        anl     c1c, #0f7h      ;CRST=0
        orl     c1c, #01h       ;SWINT=1
s_swint1:
        mov     a, c1c
        jnb     acc.0, s_swint1 ;SWINT=1?
        mov     a, #03h         ;CAN1 bus timing
        mov     dptr, #C1BT0
        movx    @dptr, a
        mov     a, #1ch
        mov     dptr, #C1BT1
        movx    @dptr, a
        mov     a, #0ffh ;CAN1SGM = 1111_1111_0000_0000
        mov     dptr, #C1SGM0
        movx    @dptr, a
        cpl     a
        mov     dptr, #C1SGM1
        movx    @dptr, a
        mov     a, #00000010b  ;CAN1MC1 = rx, 11-bit ID, IDmask
        mov     dptr, #C1M1F
        movx    @dptr, a
        mov     a, #0c3h        ;receiver ID=1100_0011_1111_0000
        mov     dptr, #C1M1AR0
        movx    @dptr, a
        mov     a, #0f0h
        mov     dptr, #C1M1AR1
        movx    @dptr, a
        anl     c1c, #0feh      ;SWINT=0
c_swint1:
        mov     a, c1c
        jb      acc.0, c_swint1 ;SWINT=0?
        mov     c1m1c, #80h     ;enable receiver
;-------------------------------------------------
;CAN0 configuration
;
;CxBT0.7-6 = Tsjw = 1,2,3,4 Tqu
;CxBT0.5-0 = BPR = 1...64
;CxBT1.7   = SMP = 1:3samp; 0:1samp
```

```
;CxBT1.6-4 = Tseg2 = x,2-8
;CxBT1.3-0 = Tseg1 = x,2-16
;---------------------------------------------
        mov     ta, #0aah
        mov     ta, #55h
        anl     c0c, #0f7h      ;CRST=0
        orl     c0c, #01h       ;SWINT=1
s_swint0:
        mov     a, c0c
        jnb     acc.0, s_swint0 ;SWINT=1?
        mov     a, #03h         ;CAN0 bus timing
        mov     dptr, #C0BT0
        movx    @dptr, a
        mov     a, #1Ch
        mov     dptr, #C0BT1
        movx    @dptr, a
        mov     a, #18h         ;CAN0MC1 = tx, 11-bit ID, 1dtbyc
        mov     dptr, #C0M1F
        movx    @dptr, a
        mov     a, #0C3h        ;ID=1100_0011_0011_1100
        mov     dptr, #C0M1AR0
        movx    @dptr, a
        cpl     a
        mov     dptr, #C0M1AR1
        movx    @dptr, a
        mov     a, #01h         ;one data byte = 01h
        mov     dptr, #C0M1D0
        movx    @dptr, a
        anl     c0c, #0feh
c_swint0:
        mov     a, c0c
        jb      acc.0, c_swint0 ;SWINT=0?
        mov     c0m1c, #85h     ;enable transmitter
        mov r1, #0ffh
        mov r0, #0ffh
        djnz r1,$
        djnz r0,$
        jmp     start
end
```

Software Final

Ficheiro de definições center.h:

```
can_object volatile xdata    *buffer= 0x00010;

can_object  volatile xdata   *CENTER_0_MESSAGE_1=   0x00EE12;
can_object  volatile xdata   *CENTER_0_MESSAGE_2=   0x00EE22;
can_object  volatile xdata   *CENTER_0_MESSAGE_3=   0x00EE32;
can_object  volatile xdata   *CENTER_0_MESSAGE_4=   0x00EE42;
can_object  volatile xdata   *CENTER_0_MESSAGE_5=   0x00EE52;
can_object  volatile xdata   *CENTER_0_MESSAGE_6=   0x00EE62;
can_object  volatile xdata   *CENTER_0_MESSAGE_7=   0x00EE72;
can_object  volatile xdata   *CENTER_0_MESSAGE_8=   0x00EE82;
can_object  volatile xdata   *CENTER_0_MESSAGE_9=   0x00EE92;
can_object  volatile xdata   *CENTER_0_MESSAGE_10=  0x00EEA2;
can_object  volatile xdata   *CENTER_0_MESSAGE_11=  0x00EEB2;
can_object  volatile xdata   *CENTER_0_MESSAGE_12=  0x00EEC2;
can_object  volatile xdata   *CENTER_0_MESSAGE_13=  0x00EED2;
can_object  volatile xdata   *CENTER_0_MESSAGE_14=  0x00EEE2;
can_object  volatile xdata   *CENTER_0_MESSAGE_15=  0x00EEF2;
```

Ficheiro de definições error.h:

```
//Mask status

#define BUS_OFF 0x80
#define ERR_WRN 0x40
#define ERR_CODE 0x07

//Interrupt register

#define NO_PENDING_INTERRUPT            0
#define CHANGE_CAN_STATUS_REGISTER      1
#define MESSAGE_15                      2
#define MESSAGE_1                       3
#define MESSAGE_2                       4
#define MESSAGE_3                       5
#define MESSAGE_4                       6
#define MESSAGE_5                       7
#define MESSAGE_6                       8
#define MESSAGE_7                       9
#define MESSAGE_8                       10
#define MESSAGE_9                       11
#define MESSAGE_10                      12
#define MESSAGE_11                      13
#define MESSAGE_12                      14
#define MESSAGE_13                      15
#define MESSAGE_14                      16
```

Ficheiro de definições movx.h:

```
#define C0MID0c          *(unsigned char xdata*)      0x0EE00
#define C0MA0c           *(unsigned char xdata*)      0x0EE01
#define C0MID1c          *(unsigned char xdata*)      0x0EE02

//this file works only if CMA bit is set to 0
//(MCON.5) CMA=0, xxxx=00EE; CMA=1, xxxx=4010.

#define C0MA1c           *(unsigned char xdata*)      0x0EE03

#define C0SGM0c          *(unsigned char xdata*)      0x0EE06
#define C0SGM1c          *(unsigned char xdata*)      0x0EE07
#define C0EGM0c          *(unsigned char xdata*)      0x0EE08
#define C0EGM1c          *(unsigned char xdata*)      0x0EE09
#define C0EGM2c          *(unsigned char xdata*)      0x0EE0A
#define C0EGM3c          *(unsigned char xdata*)      0x0EE0B
#define C0M15M0c         *(unsigned char xdata*)      0x0EE0C
#define C0M15M1c         *(unsigned char xdata*)      0x0EE0D
#define C0M15M2c         *(unsigned char xdata*)      0x0EE0E
#define C0M15M3c         *(unsigned char xdata*)      0x0EE0F

#define C1MID0c          *(unsigned char xdata*)      0x0EF00
#define C1MA0c           *(unsigned char xdata*)      0x0EF01
#define C1MID1c          *(unsigned char xdata*)      0x0EF02
#define C1MA1c           *(unsigned char xdata*)      0x0EF04
#define C1BT1c           *(unsigned char xdata*)      0x0EF05
#define C1SGM0c          *(unsigned char xdata*)      0x0EF06
#define C1SGM1c          *(unsigned char xdata*)      0x0EF07
#define C1EGM0c          *(unsigned char xdata*)      0x0EF08
#define C1EGM1c          *(unsigned char xdata*)      0x0EF09
#define C1EGM2c          *(unsigned char xdata*)      0x0EF0A
#define C1EGM3c          *(unsigned char xdata*)      0x0EF0B
#define C1M15M0c         *(unsigned char xdata*)      0x0EF0C
#define C1M15M1c         *(unsigned char xdata*)      0x0EF0D
#define C1M15M2c         *(unsigned char xdata*)      0x0EF0E
#define C1M15M3c         *(unsigned char xdata*)      0x0EF0F

#define C0M1AR0c         *(unsigned char xdata*)      0x0EE12
#define C0M1AR1c         *(unsigned char xdata*)      0x0EE13
#define C0M1AR2c         *(unsigned char xdata*)      0x0EE14
#define C0M1AR3c         *(unsigned char xdata*)      0x0EE15
#define C0M1Fc           *(unsigned char xdata*)      0x0EE16
#define C0M1D0c          *(unsigned char xdata*)      0x0EE17
#define C0M1D1c          *(unsigned char xdata*)      0x0EE18
#define C0M1D2c          *(unsigned char xdata*)      0x0EE19
#define C0M1D3c          *(unsigned char xdata*)      0x0EE1A
#define C0M1D4c          *(unsigned char xdata*)      0x0EE1B
#define C0M1D5c          *(unsigned char xdata*)      0x0EE1C
#define C0M1D6c          *(unsigned char xdata*)      0x0EE1D
#define C0M1D7c          *(unsigned char xdata*)      0x0EE1E

#define C0M14AR0c        *(unsigned char xdata*)      0x0EEE2
#define C0M14AR1c        *(unsigned char xdata*)      0x0EEE3
#define C0M14AR2c        *(unsigned char xdata*)      0x0EEE4
#define C0M14AR3c        *(unsigned char xdata*)      0x0EEE5
```

```
#define C0M2AR0c        *(unsigned char xdata*)     0x0EE22
#define C0M2AR1c        *(unsigned char xdata*)     0x0EE23
#define C0M2AR2c        *(unsigned char xdata*)     0x0EE24
#define C0M2AR3c        *(unsigned char xdata*)     0x0EE25

#define C0M2Fc          *(unsigned char xdata*)     0x0EE26

#define C0M2D0c         *(unsigned char xdata*)     0x0EE27
#define C0M2D1c         *(unsigned char xdata*)     0x0EE28
#define C0M2D2c         *(unsigned char xdata*)     0x0EE29
#define C0M2D3c         *(unsigned char xdata*)     0x0EE2A
#define C0M2D4c         *(unsigned char xdata*)     0x0EE2B
#define C0M2D5c         *(unsigned char xdata*)     0x0EE2C
#define C0M2D6c         *(unsigned char xdata*)     0x0EE2D
#define C0M2D7c         *(unsigned char xdata*)     0x0EE2E
```

Ficheiro de definições types.h:

```
//Type definitions

//Structs definitions

typedef struct
{
        unsigned char cmar0;
        unsigned char cmar1;
        unsigned char cmar2;
        unsigned char cmar3;
        unsigned char cmf;
        unsigned char packet[8];
}can_object;


typedef unsigned char BYTE;
typedef unsigned int  UINT;

//Constant definitions

#define TRUE 1
#define FALSE 0
```

Ficheiro do programa principal:

```
#include"types.h"
#include"movx.h"
#include"center.h"
#include"error.h"


sfr C0M1CC=0xAB;
sfr C0M2CC=0xAC;
sfr C0M3CC=0xAD;
sfr C0M4CC=0xAE;
sfr C0M5CC=0xAF;
sfr C0M6CC=0xB3;

sfr ACC=0xe0;
sfr EIE=0xE8;
sfr C0C=0xA3;
sfr IE=0xA8;
sbit EA=IE^7;

sfr C0S=0xA4;
sfr C0IR=0xA5;

sfr P1=0x90;
sfr P3=0xB0;
sfr P4=0x80;

sfr SCON1=0xC0;
sfr SBUF1=0xC1;
sfr P4CNT=0x92;
sfr MCON=0xc6;
sfr ACON=0x9d;


sbit            dir             = P1^0;
sbit            rst             = P1^5;
sbit            sync            = P1^4;
sbit            cs              = P1^1;
sbit            rdy             = P3^2;

sbit            teste           =P1^6;

sbit TI1=SCON1^1;
sbit RI1=SCON1^0;
sbit REN1=SCON1^4;


unsigned char
i,aux_1,intval,intcos,tx_error,new_message,get_digit,rec_digit,reset_code,update;
can_object volatile  *reset;
can_object volatile  *transmit;
can_object volatile  *receive;
can_object volatile  *transmit_digit;
can_object volatile  *receive_digit;
can_object volatile  *update_digit;
```

```
can_object volatile  *buff_r;

unsigned char serial_send,aux,intval,intcos,tx_error,i;
unsigned char amostra[3];

void init_mem(void)
        {
                #pragma asm
                        push ACC

                        mov ACC, P4CNT
                        anl ACC, #085h /*Select 32Kb, CE0, CE1*/
                        mov TA, #0aah  /*Disable time access protection*/
                        mov TA, #055h  /*Disable time access protection*/
                        mov P4CNT, ACC

                        mov TA, #0aah  /*Disable time access protection*/
                        mov TA, #055h  /*Disable time access protection*/
                        mov MCON, #010h       /*No super node, CMA=0, no PDCE, CE
merged*/

                        mov TA, #0aah  /*Disable time access protection*/
                        mov TA, #055h  /*Disable time access protection*/
                        mov ACON, #0F8h /*16 bit addressing mode, traditional 80c51 stack
pointer*/

                        mov ACC, P5CNT
                        orl ACC, #007h  /*Preserve Bits 7..3 of P5CNT and put bits 0..2
high*/
                        mov TA, #0aah  /*Disable time access protection*/
                        mov TA, #55h   /*Disable time access protection*/
                        mov P5CNT, ACC

                        pop ACC
                #pragma endasm
}



void init_serial_1(void)
        {
                #pragma asm
                        push ACC
                        clr EA                  /*Disable Interrupts*/
                        mov ACC,P5CNT
                        anl ACC, #0DFh          //Serial port 1 routed to P1.2 and P1.3

                        mov TA, #0aah  /*Disable time access protection*/
                        mov TA, #055h  /*Disable time access protection*/
                        mov P5CNT, ACC

                        pop ACC
                #pragma endasm


        }
```

```
void wait_not_redy(void)
{
        while(~rdy);
}



void wait_serial_1 (void)
{
        while(rdy);
}

void serial_1_send (unsigned char msg)
{
        REN1=FALSE;
        dir=TRUE;
        SBUF1=(msg);
        while(~TI1); //wait end of transmition
        TI1=0;
}


void init_ad(void)
{
        sync=TRUE;
        REN1=FALSE;
        cs=FALSE;
        dir=TRUE;                                       //uC->AD

        rst = FALSE;
        for (i=0;i<255;i++);            // reset delay
        rst = TRUE;


        serial_send=0xC0;
        serial_1_send(serial_send);

        serial_send=0xC8;
        serial_1_send(serial_send);

        serial_send=0x4C;
        serial_1_send(serial_send);

        serial_send=0x40;
        serial_1_send(serial_send);

        serial_send=0x8D;
        serial_1_send(serial_send);

        serial_send=0x2E;
        serial_1_send(serial_send);

        wait_serial_1();

        serial_send=0x40;
        serial_1_send(serial_send);

        serial_send=0x89;
        serial_1_send(serial_send);
```

```
        serial_send=0x2E;
        serial_1_send(serial_send);

        wait_serial_1();

}



void read_ad(void)
{
        for (i=0;i<255;i++);                    // delay

        serial_send=0x40;
        serial_1_send(serial_send);

        serial_send=0x8A;
        serial_1_send(serial_send);

        serial_send=0x2E;
        serial_1_send(serial_send);

        wait_serial_1();  //Wait convertion

        serial_send=0x88;
        serial_1_send(serial_send);

        dir = FALSE;     //reverses direction, AD->uC
        REN1=TRUE;

        for(i=0;i<32;i++);

        for(i=0;i<3;i++) {
                while (~RI1);
                        amostra[i]=SBUF1;
                        RI1=0;
        }
        REN1=FALSE;
        dir=TRUE;
}



void load(){

        new_message=FALSE;
        C0M2CC=0;
        for(i=0;i<3;i++){
                transmit->packet[i]=amostra[i];
        }
        for(i=3;i<8;i++){
                transmit->packet[i]=0;
        }

        C0M2CC=0xC5;

        while ((C0M2CC && 0x04) == 0x04);
```

```
        C0M2CC=0x00;
        C0M1CC=0xA0;

}

void id_receive (void) {

        receive->cmf=0;
        receive->cmar0=0xC3;
        receive->cmar1=0x7C;
        C0M1CC=0xA0;

}

void id(void){

        transmit->cmf=0x88;
        transmit->cmar0=0xc3;
        transmit->cmar1=0x3c;
}

void id_reset (void) {

        reset->cmf=0;
        reset->cmar0=0xC3;
        reset->cmar1=0x00;
        C0M3CC=0xA0;

}


void id_receive_digt (void) {

        receive_digit->cmf=0;
        receive_digit->cmar0=0xC3;
        receive_digit->cmar1=0xA0;
        C0M4CC=0xA0;

}

void id_transmit_digt(void){

        transmit_digit->cmf=0x18;
        transmit_digit->cmar0=0xc3;
        transmit_digit->cmar1=0x80;
}


void id_update_digt (void) {

        update_digit->cmf=0;
        update_digit->cmar0=0xC3;
        update_digit->cmar1=0xC0;
        C0M6CC=0xA0;

}


void send_digital(void){
        get_digit=FALSE;
```

```
        C0M5CC=0x00;
        transmit_digit->packet[0]=aux;

        C0M5CC=0xC5;
        while ((C0M5CC && 0x04) == 0x04);

        C0M5CC=0x00;
        C0M4CC=0xA0;

}

void put_digital(void){
        teste=~teste;
        update=FALSE;

        P4=(P4&0x0f);
        P4=P4|((update_digit->packet[0])&0xf0);
        aux=P4;
        C0M6CC=0xA0;

}


void init(void){

receive=CENTER_0_MESSAGE_1;
transmit=CENTER_0_MESSAGE_2;
reset=CENTER_0_MESSAGE_3;
receive_digit=CENTER_0_MESSAGE_4;
transmit_digit=CENTER_0_MESSAGE_5;
update_digit=CENTER_0_MESSAGE_6;

buff_r=buffer;

#pragma asm

dpx     equ     (093h)
;*************** CAN sfrs ***************************
c0rms0          equ     (096h)
c0rms1          equ     (097h)
c0tma0          equ     (09eh)
c0tma1          equ     (09fh)

p5cnt           equ     (0a2h)
c0cx            equ     (0a3h)
;c0s             equ     (0a4h)
;c0ir            equ     (0a5h)
c0te            equ     (0a6h)
c0re            equ     (0a7h)
c0m1c           equ     (0abh)
c0m2c           equ     (0ach)
c0m3c           equ     (0adh)
c0m4c           equ     (0aeh)
c0m5c           equ     (0afh)
c0m6c           equ     (0b3h)
c0m7c           equ     (0b4h)
c0m8c           equ     (0b5h)
c0m9c           equ     (0b6h)
c0m10c          equ     (0b7h)
c0m11c          equ     (0bbh)
```

```
c0m12c          equ     (0bch)
c0m13c          equ     (0bdh)
c0m14c          equ     (0beh)
c0m15c          equ     (0bfh)
ta              equ     (0c7h)
c1c             equ     (0e3h)
c1s             equ     (0e4h)
c1ir            equ     (0e5h)
c1te            equ     (0e6h)
c1re            equ     (0e7h)
c1m1c           equ     (0ebh)
c1m2c           equ     (0ech)
;*************** config registers ********************
C0MID0          equ     (0EE00h)
C0MID1          equ     (0EE02h)
C0MA0           equ     (0EE01h)
C0MA1           equ     (0EE03h)
C0BT0           equ     (0EE04h)
C0BT1           equ     (0EE05h)
C1BT0           equ     (0EF04h)
C1BT1           equ     (0EF05h)
C0SGM0          equ     (0EE06h)
C0SGM1          equ     (0EE07h)
C0EGM0          equ     (0EE08h)
C1SGM0          equ     (0EF06h)
C1SGM1          equ     (0EF07h)
C0M1AR0         equ     (0EE12h)
C0M1AR1         equ     (0EE13h)
C0M1AR2         equ     (0EE14h)
C0M1F           equ     (0EE16h)
C0M1D0          equ     (0EE17h)
C0M1D1          equ     (0EE18h)
C0M1D2          equ     (0EE19h)
C0M1D3          equ     (0EE1ah)
C0M1D4          equ     (0EE1bh)
C0M1D5          equ     (0EE1ch)
C0M1D6          equ     (0EE1dh)
C0M1D7          equ     (0EE1eh)
C1M1AR0          equ    (0EF12h)
C1M1AR1         equ     (0EF13h)
C1M1F           equ     (0EF16h)
C1M1D0          equ     (0EF17h)


;-----------------------------------------------
; main routine
;
;-----------------------------------------------
start:
        ;mov    sp, #40h        ;mov stack start
        mov     dpx, #00h       ;CAN MOVX = 00EE, 00EF
        mov     ta, #0aah
        mov     ta, #55h
        orl     p5cnt, #08h     ;enable CAN0 and CAN1
;-----------------------------------------------
;CAN0 configuration
;
;CxBT0.7-6 = Tsjw = 1,2,3,4 Tqu
;CxBT0.5-0 = BPR = 1...64
;CxBT1.7   = SMP = 1:3samp; 0:1samp
```

```
;CxBT1.6-4 = Tseg2 = x,2-8
;CxBT1.3-0 = Tseg1 = x,2-16
;---------------------------------------------
        mov     ta, #0aah
        mov     ta, #55h
        anl     c0cx, #0f7h     ;CRST=0
        orl     c0cx, #01h      ;SWINT=1
s_swint0:
        mov     a, c0cx
        jnb     acc.0, s_swint0 ;SWINT=1?
        mov     a, #0C3h                ;CAN0 bus timing
        mov     dptr, #C0BT0
        movx    @dptr, a
        mov     a, #3Ch
        mov     dptr, #C0BT1
        movx    @dptr, a
                anl     c0cx, #0feh
c_swint0:
        mov     a, c0cx
        jb      acc.0, c_swint0 ;SWINT=0?



#pragma endasm


}

void init_can_interrup(void){
        EA=FALSE;                                       //Disable all interrupts
        EIE=0x40;                                       //Enable CAN0 interrupt
        C0C=0xC0;                                       //Enable ERROR and Status
interrupt
        EA=TRUE;
}



void main(void){

init();
init_mem();
init_can_interrup();
id_receive();
id();
id_reset();
id_transmit_digt();
id_receive_digt();
id_update_digt();

        while(1){


                if (new_message) {
                        init_ad();
                        read_ad();
                        load();
                }
                if (reset_code) {
                        reset_code=FALSE;
```

```
                        teste=~teste;
                        C0M3CC=0xA0;

                }
                if(get_digit){
                        send_digital();
                }
                if(update){
                        put_digital();
                }
        }
}


void CAN_init (void) interrupt 13
{
        intval=C0IR;
        intcos=C0S;

        if (C0M3CC & 0x01)
        {
                C0M3CC=0;
                reset_code=TRUE;
                return;
        };

        if (C0M1CC & 0x01)
        {
                new_message=TRUE;
                C0M1CC=0;
                return;
        };

        if (C0M4CC & 0x01)
        {
                get_digit=TRUE;
                C0M4CC=0;
                return;
        };

        if (C0M6CC & 0x01)
        {
                update=TRUE;
                C0M6CC=0;
                return;
        };


        if (intval==CHANGE_CAN_STATUS_REGISTER)
        {
                if (intcos & ERR_CODE)
                {
                        tx_error = TRUE;
                        C0M2CC=0x00;                    //Disable transmition
                        return;
                }

                if (intcos & BUS_OFF)
                {
                        tx_error = TRUE;
```

```
                        C0M2CC=0x00;                    //Disable transmition
                        return;
                }

                if (intcos & ERR_WRN)
                {
                        tx_error = TRUE;
                        C0M2CC=0x00;                    //Disable transmition
                        return;
                }
        return;
}
}
```

Software de Boot Loader

```
#include <REG390.H>


;=========================================================================
;                               Definitions
;=========================================================================



LF              EQU     0Ah ;           Line Feed character.
CR              EQU     0Dh ;           Carriage Return character.
ESC             EQU     1Bh ;           Escape character.
StartChar       EQU     ':' ;           Line start character for hex file.
Slash           EQU     '/' ;       Go command character.
Skip            EQU      13 ;           Value for "Skip" state.
Ch              DATA    0Fh ;           Last character received.
State           DATA    10h ;           Identifies the state in process.
DataByte        DATA    11h ;           Last data byte received.
ByteCount       DATA    12h ;           Data byte count from current line.
HighAddr        DATA    13h ;           High and low address bytes from the
LowAddr         DATA    14h ;           current data line.
RecType         DATA    15h ;           Line record type for this line.
ChkSum                  DATA    16h ;           Calculated checksum received.
HASave                  DATA    17h ;           Saves the high and low address bytes
LASave                  DATA    18h ;           from the last data line.
FilChkHi        DATA    19h ;           File checksum high byte.
FilChkLo        DATA    1Ah ;           File checksum low byte.
Flags           DATA    20h ;           State condition flags.
HexFlag         BIT     Flags.0 ;       Hex character found.
EndFlag         BIT     Flags.1 ;       End record found.
DoneFlag        BIT     Flags.2 ;       Processing done (end record or some; kind of error.
EFlags          DATA 21h ;              Exception flags.
ErrFlag1        BIT EFlags.0 ;          Non-hex character embedded in data.
ErrFlag2        BIT EFlags.1 ;          Bad record type.
ErrFlag3        BIT EFlags.2 ;          Bad line checksum.
ErrFlag4        BIT EFlags.3 ;          No data found.
ErrFlag5        BIT EFlags.4 ;          Incremented address overflow.
ErrFlag6        BIT EFlags.5 ;          Data storage verify error.
DatSkipFlag     BIT Flags.3 ;           Any data found should be ignored.
```

```
/*SFR definitions*/

;=====================================================================
; Reset and Interrupt Vectors
;=====================================================================
; The following are dummy labels for re-mapped interrupt vectors. The
; addresses should be changed to match the memory map of the target system.
; MODIFICADO PARA O DS80C390

ExInt0          EQU 8003h ;             Remap address for ext interrupt 0.
T0Int           EQU 800Bh ;             Timer 0 interrupt.
ExInt1          EQU 8013h ;             External interrupt 1.
T1Int           EQU 801Bh ;             Timer 1 interrupt.
SerInt          EQU 8023h ;             Serial port interrupt.
SerInt1         EQU 803Bh ;             Serial 1 port interrupt.
C0I             EQU 80CBh ;             CAN 0 Interrupt
C1I             EQU 8073h ;             CAN 1 Interrupt
WDTI            EQU 8063h ;             Watchdog timer
CANBUS              EQU 807Bh ;             Can 0/1 Bus activity

ORG 0000h

LJMP Start ;                            Go to the downloader program.
; The following are intended to allow re-mapping the interrupt vectors to the
; users downloaded program. The jump addresses should be adjusted to reflect
; the memory mapping used in the actual application.
; Other (or different) interrupt vectors may need to be added if the target
; processor is not an 80C51.

ORG 0003h
LJMP            ExInt0 ;                External interrupt 0.
RETI
ORG 000Bh
LJMP            T0Int ;                 Timer 0 interrupt.
RETI
ORG 0013h
LJMP ExInt1 ;                           External interrupt 1.
RETI
ORG 001Bh
LJMP T1Int ;                            Timer 1 interrupt.
RETI
ORG 0023h
LJMP SerInt ;                           Serial port interrupt.
RETI

ORG 003Bh
LJMP  SerInt1 ;
RETI
ORG 00CBh
LJMP  C0I ;
RETI
ORG 0073h
LJMP  C1I ;
RETI
ORG 0063h
LJMP  WDTI ;
RETI
ORG 007Bh
LJMP  CANBUS ;
```

```
RETI
;=====================================================================
;                       Reset and Interrupt Vectors
;=====================================================================
Start:

;Modificado para o ds80c390 P5cnt e P4cnt


                        mov ACC, P4CNT
                        anl ACC, #085h /*Select 32Kb, CE0, CE1*/
                        mov TA, #0aah  /*Disable time access protection*/
                        mov TA, #055h  /*Disable time access protection*/
                        mov P4CNT, ACC

                        mov TA, #0aah  /*Disable time access protection*/
                        mov TA, #055h  /*Disable time access protection*/
                        mov MCON, #010h       /*No super node, CMA=0, no PDCE, CE
merged*/

                        mov TA, #0aah  /*Disable time access protection*/
                        mov TA, #055h  /*Disable time access protection*/
                        mov ACON, #0F8h /*16 bit addressing mode, traditional 80c51 stack
pointer*/

                        mov ACC, P5CNT
                        orl ACC, #007h  /*Preserve Bits 7..3 of P5CNT and put bits 0..2
high*/
                        mov TA, #0aah  /*Disable time access protection*/
                        mov TA, #55h   /*Disable time access protection*/
                        mov P5CNT, ACC



MOV IE,#0 ;                     Turn off all interrupts.
MOV SP,#5Fh ;                           Start stack near top of '51 RAM.
ACALL SerStart ;                Setup and start serial port.
ACALL CRLF ;                    Send a prompt that we are here.
MOV A,#'=' ; "<CRLF> ="
ACALL PutChar
ACALL HexIn ;                   Try to read hex file from serial port.
ACALL ErrPrt ;                  Send a message for any errors or
; warnings that were noted.
MOV A,EFlags ;                          We want to get stuck if a fatal
JZ HexOK ;                      error occurred.
ErrLoop: MOV A,#'?' ;           Send a prompt to confirm that we
ACALL PutChar ;                 are 'stuck'. " ? "
ACALL GetChar ;                 Wait for escape char to flag reload.
SJMP ErrLoop
HexOK: MOV EFlags,#0 ;                  Clear errors flag in case we re-try.
ACALL GetChar ;                 Look for GO command.
CJNE A,#Slash,HexOK ;                   Ignore other characters received.
ACALL GetByte ;                 Get the GO high address byte.
JB ErrFlag1,HexOK ;             If non-hex char found, try again.
MOV HighAddr,DataByte ;         Save upper GO address byte.
ACALL GetByte ;                 Get the GO low address byte.
JB ErrFlag1,HexOK ;             If non-hex char found, try again.
MOV LowAddr,DataByte ;                  Save the lower GO address byte.
ACALL GetChar ;                 Look for CR.
CJNE A,#CR,HexOK ;              Re-try if CR not there.
```

```
; All conditions are met, so hope the data file and the GO address are all
; correct, because now we're committed.
MOV A,#'@' ;                    Send confirmation to GO. " @ "
ACALL PutChar
JNB TI,$ ;                      Wait for completion before GOing.
PUSH LowAddr ;                          Put the GO address on the stack,
PUSH HighAddr ;                 so we can Return to it.
RET ;                           Finally, go execute the user program!
;=======================================================================
;               Hexadecimal File Input Routine
;=======================================================================
HexIn: CLR A ;                  Clear out some variables.
MOV State,A
MOV Flags,A
MOV HighAddr,A
MOV LowAddr,A
MOV HASave,A
MOV LASave,A
MOV ChkSum,A
MOV FilChkHi,A
MOV FilChkLo,A
MOV EFlags,A
SETB ErrFlag4 ;                 Start with a 'no data' condition.
StateLoop: ACALL GetChar ;      Get a character for processing.
ACALL AscHex ;                          Convert ASCII-hex character to hex.
MOV Ch,A ;                      Save result for later.
ACALL GoState ;                 Go find the next state based on
; this char.
JNB DoneFlag,StateLoop ;        Repeat until done or terminated.
ACALL PutChar ;                 Send the file checksum back as
MOV A,#'(' ;                    confirmation. " (abcd) "
ACALL PutChar
MOV A,FilChkHi
ACALL PrByte
MOV A,FilChkLo
ACALL PrByte
MOV A,#')'
ACALL PutChar
ACALL CRLF
RET ;                           Exit to main program.
; Find and execute the state routine pointed to by "State".
GoState: MOV A,State ;                  Get current state.
ANL A,#0Fh ;                    Insure branch is within table range.
RL A ;                          Adjust offset for 2 byte insts.
MOV DPTR,#StateTable
JMP @A+DPTR ;                           Go to appropriate state.
StateTable: AJMP StWait ;       0 - Wait for start.
AJMP StLeft ;                   1 - First nibble of count.
AJMP StGetCnt ;                 2 - Get count.
AJMP StLeft ;                   3 - First nibble of address byte 1.
AJMP StGetAd1 ;                 4 - Get address byte 1.
AJMP StLeft ;                   5 - First nibble of address byte 2.
AJMP StGetAd2 ;                 6 - Get address byte 2.
AJMP StLeft ;                   7 - First nibble of record type.
AJMP StGetRec ;                 8 - Get record type.
AJMP StLeft ;                   9 - First nibble of data byte.
AJMP StGetDat ;                 10 - Get data byte.
AJMP StLeft ;                   11 - First nibble of checksum.
```

```
AJMP StGetChk ;                 12 - Get checksum.
AJMP StSkip ;                   13 - Skip data after error condition.
AJMP BadState ;                 14 - Should never get here.
AJMP BadState ;                 15 - " " " "


; This state is used to wait for a line start character. Any other characters
; received prior to the line start are simply ignored.
StWait: MOV A,Ch ; Retrieve input character.
CJNE A,#StartChar,SWEX ;        Check for line start.
INC State ;                     Received line start.
SWEX: RET
; Process the first nibble of any hex byte.
StLeft: MOV A,Ch ;              Retrieve input character.
JNB HexFlag,SLERR ;             Check for hex character.
ANL A,#0Fh ;                    Isolate one nibble.
SWAP A ;                        Move nibble too upper location.
MOV DataByte,A ;                Save left/upper nibble.
INC State ;                     Go to next state.
RET ;                           Return to state loop.
SLERR: SETB ErrFlag1 ;                  Error - non-hex character found.
SETB DoneFlag ;                 File considered corrupt. Tell main.
RET
; Process the second nibble of any hex byte.
StRight: MOV A,Ch ;             Retrieve input character.
JNB HexFlag,SRERR ;             Check for hex character.
ANL A,#0Fh ;                    Isolate one nibble.
ORL A,DataByte ;                Complete one byte.
MOV DataByte,A ;                Save data byte.
ADD A,ChkSum ;                          Update line checksum,
MOV ChkSum,A ;                          and save.
RET ;                           Return to state loop.
SRERR: SETB ErrFlag1 ;                  Error - non-hex character found.
SETB DoneFlag ;                 File considered corrupt. Tell main.
RET
; Get data byte count for line.
StGetCnt: ACALL StRight ;       Complete the data count byte.
MOV A,DataByte
MOV ByteCount,A
INC State ;                     Go to next state.
RET ;                           Return to state loop.
; Get upper address byte for line.
StGetAd1: ACALL StRight ;       Complete the upper address byte.
MOV A,DataByte
MOV HighAddr,A ;                Save new high address.
INC State ;                     Go to next state.
RET ;                           Return to state loop.
; Get lower address byte for line.
StGetAd2: ACALL StRight ;       Complete the lower address byte.
MOV A,DataByte
MOV LowAddr,A ;                 Save new low address.
INC State ;                     Go to next state.
RET ;                           Return to state loop.
; Get record type for line.
StGetRec: ACALL StRight ;       Complete the record type byte.
MOV A,DataByte
MOV RecType,A ;                 Get record type.
JZ SGRDat ;                     This is a data record.
CJNE A,#1,SGRErr ;              Check for end record.
SETB EndFlag ;                          This is an end record.
```

```
SETB DatSkipFlag ;              Ignore data embedded in end record.
MOV State,#11 ;                 Go to checksum for end record.
SJMP SGREX


SGRDat:                         INC State ; Go to next state.
SGREX: RET ;                    Return to state loop.
SGRErr: SETB ErrFlag2 ;         Error, bad record type.
SETB DoneFlag ;                 File considered corrupt. Tell main.
RET
; Get a data byte.
StGetDat: ACALL StRight ;       Complete the data byte.
JB DatSkipFlag,SGD1 ;           Don't process the data if the skip
; flag is on.
ACALL Store ;                   Store data byte in memory.
MOV A,DataByte ;                Update the file checksum,
ADD A,FilChkLo ;                which is a two-byte summation of
MOV FilChkLo,A ;                all data bytes.
CLR A
ADDC A,FilChkHi
MOV FilChkHi,A
MOV A,DataByte
SGD1: DJNZ ByteCount,SGDEX ;        Last data byte?
INC State ;                     Done with data, go to next state.
SJMP SGDEX2
SGDEX: DEC State ;              Set up state for next data byte.
SGDEX2: RET ;                       Return to state loop.
; Get checksum.
StGetChk: ACALL StRight ;       Complete the checksum byte.
JNB EndFlag,SGC1 ;              Check for an end record.
SETB DoneFlag ;                 If this was an end record,
SJMP SGCEX ;                        we are done.
SGC1: MOV A,ChkSum ;                Get calculated checksum.
JNZ SGCErr ;                    Result should be zero.
MOV ChkSum,#0 ;                 Preset checksum for next line.
MOV State,#0 ;                  Line done, go back to wait state.
MOV LASave,LowAddr ;                Save address byte from this line for
MOV HASave,HighAddr ;               later check.
SGCEX: RET ;                    Return to state loop.
SGCErr: SETB ErrFlag3 ;         Line checksum error.
SETB DoneFlag ;                 File considered corrupt. Tell main.
RET
; This state used to skip through any additional data sent, ignoring it.
StSkip: RET ;                   Return to state loop.
; A place to go if an illegal state comes up somehow.
BadState: MOV State,#Skip ;     If we get here, something very bad
RET ;                           happened, so return to state loop.
; Store - Save data byte in external RAM at specified address.
Store: MOV DPH,HighAddr ;       Set up external RAM address in DPTR.
MOV DPL,LowAddr
MOV A,DataByte
MOVX @DPTR,A ;                      Store the data.
MOVX A,@DPTR ;                      Read back data for integrity check.
CJNE A,DataByte,StoreErr ;      Is read back OK?
CLR ErrFlag4 ;                  Show that we've found some data.
INC DPTR ;                      Advance to the next addr in sequence.
MOV HighAddr,DPH ;              Save the new address
MOV LowAddr,DPL
CLR A
CJNE A,HighAddr,StoreEx ;       Check for address overflow
```

```
CJNE A,LowAddr,StoreEx ;        (both bytes are 0).
SETB ErrFlag5 ;                 Set warning for address overflow.


StoreEx: RET
StoreErr: SETB ErrFlag6 ;       Data storage verify error.
SETB DoneFlag ;                 File considered corrupt. Tell main.
RET
;=====================================================================
;                           Subroutines
;=====================================================================
; Subroutine summary:
; SerStart - Serial port setup and start.
; GetChar - Get a character from the serial port for processing.
; GetByte - Get a hex byte from the serial port for processing.
; PutChar - Output a character to the serial port.
; AscHex - See if char in ACC is ASCII–hex and if so convert to hex nibble.
; HexAsc - Convert a hexadecimal nibble to its ASCII character equivalent.
; ErrPrt - Return any error codes to our host.
; CRLF - output a carriage return / line feed pair to the serial port.
; PrByte - Send a byte out the serial port in ASCII hexadecimal format.
; SerStart - Serial port setup and start.
SerStart: MOV A,PCON ; Make sure SMOD is off.
CLR ACC.7
MOV PCON,A
MOV TH1,#0FDh ;             Set up timer 1.
MOV TL0,#0FDh
MOV TMOD,#20h
MOV TCON,#40h
MOV SCON,#52h ;             Set up serial port.
RET
; GetByte -                 Get a hex byte from the serial port for processing.
GetByte: ACALL GetChar ;    Get first character of byte.
ACALL AscHex ;                  Convert to hex.
MOV Ch,A ;                  Save result for later.
ACALL StLeft ;              Process as top nibble of a hex byte.
ACALL GetChar ;             Get second character of byte.
ACALL AscHex ;                  Convert to hex.
MOV Ch,A ;                  Save result for later.
ACALL StRight ;             Process as bottom nibble of hex byte.
RET
; GetChar - Get a character from the serial port for processing.
GetChar: JNB RI,$ ;         Wait for receiver flag.
CLR RI ;                    Clear receiver flag.
MOV A,SBUF ;                Read character.
CJNE A,#ESC,GCEX ;          Re-start immediately if Escape char.
LJMP Start
GCEX: RET
; PutChar - Output a character to the serial port.
PutChar: JNB TI,$ ;         Wait for transmitter flag.
CLR TI ;                    Clear transmitter flag.
MOV SBUF,A ;                Send character.
RET
; AscHex - See if char in ACC is ASCII-hex and if so convert to a hex nibble.
; Returns nibble in A, HexFlag tells if char was really hex. The ACC is not
; altered if the character is not ASCII hex. Upper and lower case letters
; are recognized.


AscHex: CJNE A,#'0',AH1 ;   Test for ASCII numbers.
```

```
AH1: JC AHBad ;             Is character is less than a '0'?
CJNE A,#'9'+1,AH2 ;         Test value range.
AH2: JC AHVal09 ;           Is character is between '0' and '9'?
CJNE A,#'A',AH3 ;           Test for upper case hex letters.
AH3: JC AHBad ;             Is character is less than an 'A'?
CJNE A,#'F'+1,AH4 ;         Test value range.
AH4: JC AHValAF ;           Is character is between 'A' and 'F'?
CJNE A,#'a',AH5 ;           Test for lower case hex letters.
AH5: JC AHBad ;             Is character is less than an 'a'?
CJNE A,#'f'+1,AH6 ;         Test value range.
AH6: JNC AHBad ;            Is character is between 'a' and 'f'?
CLR C
SUBB A,#27h ;               Pre-adjust character to get a value.
SJMP AHVal09 ;                      Now treat as a number.
AHBad: CLR HexFlag ;        Flag char as non-hex, don't alter.
SJMP AHEX ;                 Exit
AHValAF:                    CLR C
SUBB A,#7 ;                 Pre-adjust character to get a value.
AHVal09:                    CLR C
SUBB A,#'0' ;               Adjust character to get a value.
SETB HexFlag ;                      Flag character as 'good' hex.
AHEX: RET
; HexAsc - Convert a hexadecimal nibble to its ASCII character equivalent.
HexAsc: ANL A,#0Fh ;        Make sure we're working with only
; one nibble.
CJNE A,#0Ah,HA1 ;           Test value range.
HA1: JC HAVal09 ;           Value is 0 to 9.
ADD A,#7 ;                  Value is A to F, extra adjustment.
HAVal09: ADD A,#'0' ;       Adjust value to ASCII hex.
RET
; ErrPrt - Return an error code to our host.
ErrPrt: MOV A,#':' ;        First, send a prompt that we are
CALL PutChar ;                      still here.
MOV A,EFlags ;                      Next, print the error flag value if
JZ ErrPrtEx ;               it is not 0.
CALL PrByte
ErrPrtEx:                   RET
; CRLF - output a carriage return / line feed pair to the serial port.
CRLF:                       MOV A,#CR
CALL PutChar
MOV A,#LF
CALL PutChar
RET
; PrByte - Send a byte out the serial port in ASCII hexadecimal format.
PrByte: PUSH ACC ;          Print ACC contents as ASCII hex.
SWAP A
CALL HexAsc ;               Print upper nibble.
CALL PutChar
POP ACC
CALL HexAsc ;               Print lower nibble.
CALL PutChar
RET
;=====================================================================
                END
```

Hardware

Esquemáticos