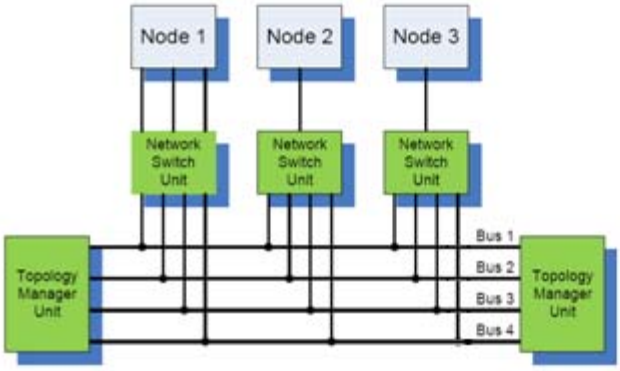
Universidade de Aveiro
2009
Departamento de Electrónica, Telecomunicações e Informática

Vítor Hugo Martins Silva

Desenvolvimento de Componentes para Sistemas de Segurança Crítica



Universidade de Aveiro
2009
Departamento de Electrónica, Telecomunicações e Informática

Vítor Hugo Martins Silva

Desenvolvimento de Componentes para Sistemas de Segurança Crítica

dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Professor Doutor Arnaldo Silva Rodrigues de Oliveira, Professor Auxiliar Convidado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro e do Mestre Valter Filipe Miranda Castelão da Silva, Assistente de 2º Triénio na Escola Superior de Tecnologia e Gestão de Águeda.

o júri

presidente

Doutor José Alberto Gouveia Fonseca
professor associado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Doutor José Carlos Meireles Monteiro Metrôlho
professor adjunto do Departamento de Engenharia Informática da Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco

Doutor Arnaldo Silva Rodrigues de Oliveira
professor auxiliar convidado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Mestre Valter Filipe Miranda Castelão da Silva
assistente de 2º Triénio na Escola Superior de Tecnologia e Gestão de Águeda

agradecimentos

Desde o inicio desta tese que tive o apoio e contribuição de várias pessoas, umas com o seu conhecimento, outras com palavras de incentivo, mas foram todas muito importantes e sem elas a conclusão deste trabalho não seria possível. Por esses motivos quero deixar aqui os devidos agradecimentos a todos eles.

Quero começar por agradecer ao Prof. Doutor José Alberto Fonseca que numa fase inicial contribuiu com o seu conhecimento e orientação. Agradeço também ao Mestre Valter Filipe Silva que em algumas fases do trabalho me ajudou a ultrapassar alguns obstáculos. Em particular gostaria de agradecer ao Prof. Doutor Arnaldo Silva Rodrigues de Oliveira que inicialmente começou como meu co-orientador e mais tarde como orientador, por todo o tempo dispendido, conhecimento partilhado, sugestões e discussões que contribuíram de uma forma muito importante na minha formação e que se reflectiu na qualidade desta dissertação.

Por último, gostaria de agradecer à minha família e amigos por todo o apoio que me deram, em especial à minha mãe que sempre me incentivou e apoiou nas alturas mais difíceis. É com muito orgulho que dedico este trabalho à minha mãe.

palavras-chave

Comunicações Industriais, Segurança Critica, FPGAs, Sistemas Digitais Reconfiguráveis, Sistemas Tempo-Real, CAN.

resumo

Hoje em dia é muito frequente a utilização de sistemas de controlo em aplicações de segurança crítica, por exemplo nos aviões, automóveis, etc. Um sistema de segurança crítica é assim designado porque em caso de falha pode resultar em grandes prejuízos monetários ou, no pior cenário, em perdas de vidas humanas. Este tipo de sistemas têm que ser robustos e tolerantes a falhas para falhar o mínimo possível e de uma forma segura. É também necessário que estes sistemas atendam a requisitos temporais para garantir o seu correcto funcionamento, sendo por isso designados de sistemas tempo-real. Quando tais sistemas são distribuídos, como em redes de sensores, actuadores e controladores interligados por barramentos de campo, a comunicação desempenha um papel importante no comportamento temporal.

Tem-se assistido nos últimos anos a um incremento da investigação e desenvolvimento da área de sistemas de segurança crítica. Têm vindo a ser criados alguns standards nesta área, sendo actualmente muito utilizado o CAN – *Controller Area Network*. No DETI tem-se realizado muita investigação em torno do CAN, tendo-se recentemente trabalhado em sistemas que permitem introduzir redundância e maior largura de banda no barramento sem modificação dos módulos a ele ligados. Os sistemas *x-by-wire* são um exemplo de sistemas de segurança crítica utilizados nos automóveis onde a questão da largura de banda é muito importante. Esta dissertação é uma continuação desse trabalho, onde se pretende implementar na prática um sistema que introduza estas características a um barramento CAN.

As principais contribuições originais desta dissertação são o desenvolvimento de alguns dos componentes projectados no trabalho de investigação em curso, nomeadamente um gestor de topologia e um comutador. Estes componentes permitem adicionar barramentos redundantes e gerir de uma forma dinâmica a topologia numa rede CAN. Para tal foram utilizadas FPGAs – *Field Programmable Gate Arrays*, um processador da Xilinx e algum hardware desenvolvido no DETI, o CLAN e o respectivo controlador que permite a sua interface com microprocessadores. A prototipagem em FPGAs facilitou e simplificou a tarefa de simulação da lógica necessária à implementação, tanto do gestor de topologia como do comutador.

keywords

Industrial Communications, Critical Security, FPGAs, Reconfigurable Digital Systems, Real Time Systems, CAN.

abstract

Nowadays, the use of control systems in critical security applications is very common, for instance in airplanes, automobiles, etc. It is called critical security application because in case any fault occurs, it can cause huge monetary damages or, in the worst scenario, it can cause the death of human lives. This type of system must be strong and fault tolerant in order to fail the least as possible and in a secure and safe way. It is also necessary that these systems answer to time requirements so that they can guarantee their proper performance, being therefore known as real-time systems. Whenever these systems are distributed, as in sensor networks, activators and controllers interconnected by fieldbuses, communication assumes an important role in time behavior.

In the past few years we have been watching to an increase in the critical security systems research and development area. Some standards have been created in this field, being CAN - Controller Area Network one of the most used in the present. In DETI, plenty of research has been made about CAN, and recently important work has been done in systems that allow the introduction of redundancy and greater bandwidth in the bus without modifying the modules which are attached to it. The systems x-by-wire are just an example of critical security systems which are used in cars, and where the matter of bandwidth is actually very important. This dissertation is therefore a continuation of that work that has been developed so far, which aims to implement, in practice, a system that is able to introduce these features in a CAN bus.

The most original and more important contributions of this dissertation are the development of some of the components projected in this work of ongoing research, namely a topology manager and a commuter. These components allow to attach redundant buses and to manage, in a dynamic kind of way, a topology in a CAN network. In order to achieve it, there have been used FPGAs – Field Programmable Gate Arrays, a processor from Xilinx and some hardware which had been developed in DETI, CLAN and the corresponding controller that allows its interface with microprocessors. The prototyping in FPGAs has facilitated and simplified the task of simulating the logic that was necessary to the implementation, whether from the topology manager whether from the commuter.

Índice

Lista de Figuras

Lista de Tabelas

Capítulo 1

1. Introdução

1.1. Enquadramento

Um sistema de controlo distribuído, tal como o próprio nome indica, possui seus elementos fisicamente distribuídos e interligados por uma rede de comunicação. Estes elementos, genericamente designados por nodos, podem estar ligados a sensores, ou actuadores, ou desempenhar funções de controlo. A comunicação e partilha de informação entre os nodos é feita por intermédio de redes de comunicação, que possuem para isso um protocolo que traduz um conjunto de regras que permitem a comunicação entre duas camadas homólogas (no mesmo nível). O modelo OSI, definido pela ISO (ISO, 1947), é um conjunto de protocolos abertos, desenvolvido para permitir a comunicação entre sistemas destinados a funcionar em rede.

Os sistemas distribuídos são frequentemente utilizados na indústria ou em sistemas embutidos que se podem encontrar nos automóveis, aviões, centrais de produção e distribuição de energia, etc. Nestes casos, os sistemas distribuídos são utilizados em aplicações de segurança crítica, uma vez que possuem restrições ao nível temporal na troca de mensagens entre os vários nodos que têm que ser cumpridas, caso contrário, pode resultar em graves danos materiais e no pior dos cenários na perda de vidas humanas.

Uma abordagem distribuída (ver Figura 1 a), possui vantagens em relação a uma abordagem centralizada (ver Figura 1 b), destacando-se a escalabilidade que permite um crescimento do sistema de acordo com as necessidades. No entanto, a distribuição das funcionalidades por vários nodos levanta também diversos desafios do ponto de vista da tolerância a falhas. Estes desafios, problemas e respectivas soluções têm sido um tópico de investigação muito importante e activo. Desafios estes, que se prendem com o facto de ser necessária uma grande

capacidade destes sistemas lidarem com alterações relevantes que podem afectar todo o seu funcionamento. Alterações relevantes compreendem desde nodos em falha, curto-circuito nos barramentos, particionamento da rede, entre outros, que podem afectar as comunicações e o correcto funcionamento do sistema. Assim, estes sistemas terão que ser robustos e flexíveis, através da utilização de mecanismos tolerantes a falhas, ao ponto de serem capazes de se adaptar a alterações na configuração para manter ou até melhorar o desempenho em cenários dinâmicos. O melhoramento de desempenho pode ser feito em sistemas de comunicação com redundância ao nível dos meios de comunicação, onde podem ser definidos barramentos dedicados à comunicação entre dois nodos, havendo assim uma melhor gestão da largura de banda.

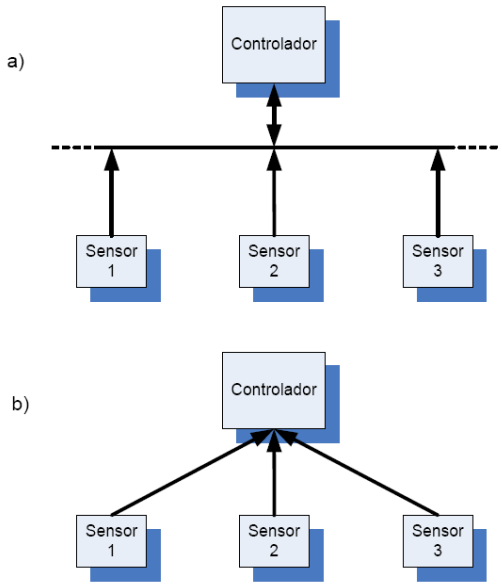


Figura 1: a) Abordagem distribuída; b) Abordagem centralizada.

Alterações na configuração podem fazer com que ocorram erros de transmissão e consequente perda de mensagens, ou até conflitos de acesso ao meio de comunicação e dessincronização entre os vários nodos. É necessária portanto uma grande coordenação entre os vários nodos do sistema, para permitir uma reconfiguração dinâmica mantendo assim o sistema em funcionamento durante e após a adaptação. Contudo, em sistemas distribuídos existem vários

pontos de falha[1] que podem comprometer a coordenação do processo de adaptação e deixar todo o sistema inconsistente. São necessários portanto mecanismos de tolerância a falhas que permitam o correcto funcionamento do sistema na presença de falhas mesmo que isso implique o seu funcionamento num modo degradado.

O CAN – *Controller Area Network* (CiA, 2001-2008) é um dos standards mais utilizados na actualidade como infra-estrutura para sistemas de comunicação baseados em barramentos partilhados. Na sua forma nativa, o CAN possui vários pontos de falha e não determinismo temporal. Por este motivo têm vindo a ser feitos esforços para o tentar melhorar tendo em vista aumentar a sua fiabilidade e assim ser possível garantir que cumpra todos os requisitos para a sua aplicação em sistemas de segurança crítica. Uma característica do CAN que é desfavorável à sua aplicação em sistemas deste tipo é o facto da transmissão de mensagens ser baseada no paradigma *event-triggered*, não havendo por isso determinismo temporal. Assim, foi feita uma extensão ao protocolo CAN original tornando-o *time-triggered* (TTCAN) (CiA, 2001-2008). Com esta alteração foi possível a sua aplicação em sistemas de tempo-real do tipo crítico, que normalmente são sistemas de segurança crítica (por exemplo *x-by-wire*), onde é necessário assegurar que todas as mensagens cumprem as suas restrições temporais mesmo quando o barramento está sobrecarregado. Como o protocolo CAN suporta comunicações *event-triggered*, a extensão TTCAN permite combinar comunicações do tipo *time-triggered* e *event-triggered* em janelas temporais distintas.

Uma das características do CAN que deprecia a sua fiabilidade é o facto da sua topologia de rede ser baseada apenas num barramento (ver Figura 2). Assim sendo, os erros propagam-se pelo meio de transmissão que é comum a toda a rede, havendo a possibilidade de um nodo em falha afectar e até mesmo bloquear

[1] Um ponto único de falha ou ponto crítico de falha é uma tradução vinda da língua inglesa da expressão *Single Point of Failure* (en:SPOF) para designar um local num sistema que, caso falhe, provoca a falha de todo o sistema.

todo o sistema. Como exemplo de falhas num barramento podemos ter cabos que se partem, curto-circuitos, nodos em falha, etc.

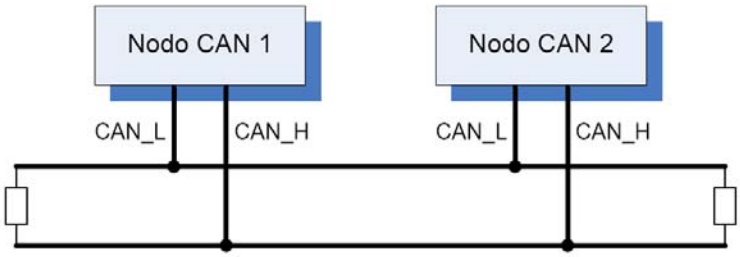


Figura 2: Barramento CAN, comunicação entre dois nodos.

Na tentativa de melhorar o CAN original tendo em vista alterar a sua topologia de rede com apenas um ponto de falha, têm vindo a ser desenvolvidas algumas soluções baseadas na replicação de barramentos. Uma solução que já havia sido adoptada pela *Ethernet* (ISO, 2003), *FlexRay* (FlexRay, 2009) e *TTP/C* (TTA-Group, 1998), é a topologia em estrela que, para além da vantagem de ser possível replicar barramentos, possui também a vantagem de evitar falhas de proximidade, pois as ligações só ficam próximas umas das outras perto do *hub* central. Nesta topologia o barramento é seccionado e todos os nodos são ligados ao *hub* central através de ligações ponto-a-ponto. Isto permite evitar que nodos em falha comprometam todas as comunicações. Esta topologia pode ser aplicada ao CAN sendo completamente compatível com todos os controladores existentes. No entanto, nesta topologia o *hub* central pode representar um ponto de falha, sendo necessário replicar este componente, quando os requisitos de segurança assim o exijam, o que faz aumentar a sua complexidade.

Uma alternativa à topologia em estrela é combinar de forma flexível a simplicidade da topologia de barramento com a fiabilidade da topologia em estrela para que seja possível a sua aplicação com facilidade a cada situação em particular. Hoje em dia é dada muita importância à fiabilidade e a elevadas larguras de banda nos barramentos, pelo que são bastante apreciadas soluções que envolvam a replicação de barramentos e que satisfaçam ambas as exigências.

Dependendo da aplicação, é possível e relativamente simples implementar características que permitam gerir tolerância a falhas e usar a largura de banda para fazer transmissões redundantes de mensagens. É possível também reagir a falhas no barramento, permitindo a continuação da comunicação através da utilização dos barramentos disponíveis em ausência de falhas. Na Figura 3 está representado um pequeno exemplo de um barramento com 2-níveis de redundância.

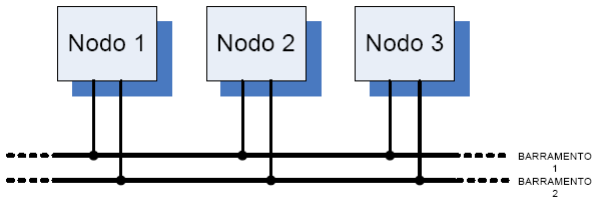


Figura 3: Barramento com 2-níveis de redundância.

É assim possível transmitir tráfego não crítico e modificar durante o decorrer da transmissão o modo de operação para modos degradados na presença de falhas no barramento. Isto é feito na infra-estrutura da rede apenas, sem modificar os nodos do sistema distribuído (Silva, et al., 2006). A implementação ao ser feita na infra-estrutura da rede facilita a aplicação do sistema sem que seja necessário substituir ou modificar os nodos já existentes, sendo apenas necessário acrescentar o hardware necessário entre os nodos e os barramentos.

1.2. Motivação

A topologia de rede em estrela possui características importantes que permitem isolar falhas. Neste tipo de topologia um elemento central controla o fluxo de dados da rede, ficando ligado ponto-a-ponto a cada nodo. É esta característica que permite isolar um nodo em falha, evitando assim que o resto da rede seja afectada, permitindo assim a continuação de um bom funcionamento da

rede. Esta é uma das razões que tornaram esta topologia muito popular nas redes de campo[2]. No entanto, esta topologia tem a desvantagem de requerer maiores comprimentos de cabo para a implementar do que numa rede baseada em barramentos (Silva, et al., 2006). Mesmo colocando todos os nodos alinhados como num barramento, as ligações necessitam de mais cablagem, estas situações estão representadas na Figura 4 e Figura 5.

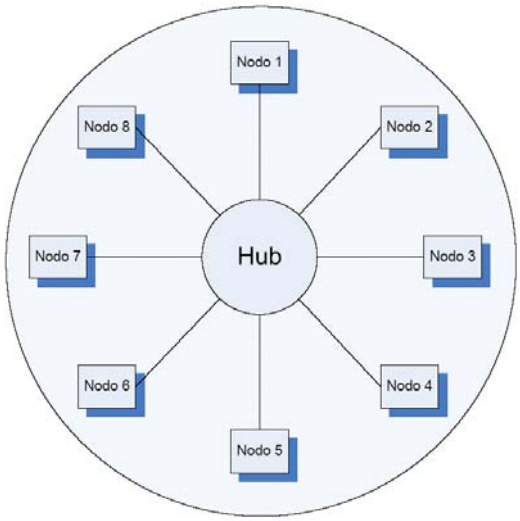


Figura 4: Topologia estrela circular - adaptado de (Silva, et al., 2006).

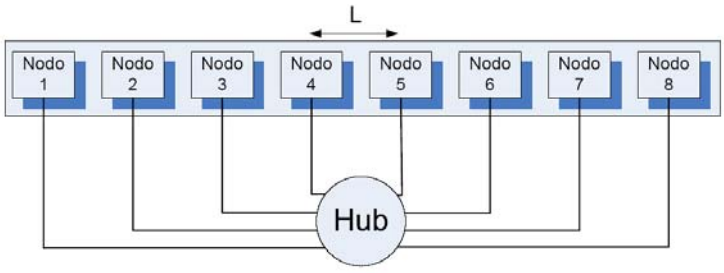


Figura 5: Topologia estrela em linha - adaptado de (Silva, et al., 2006).

A quantidade de cablagem necessária para implementar uma topologia de rede torna-se muito importante quando se pretende implementar replicação de

[2] Classificação generalista utilizada para praticamente todas as redes industriais de dados. Visam a interligação de sensores, actuadores, placas de entrada/saída e sistemas de controlo local em instalações industriais.

barramentos para adicionar capacidades de tolerância a falhas, uma vez que o custo e complexidade aumentam com este parâmetro.

No caso específico do CAN a sua limitação em termos de largura de banda tem sido motivo de preocupação para o seu uso em aplicações, como por exemplo, em sistemas *X-by-wire* (Shaheen, et al., 2003). Replicação de barramentos pode adicionar maior largura de banda e fiabilidade. Para o caso específico do CAN é possível adicionar nodos com apenas uma ligação a um barramento ou com capacidade de ligação a barramentos replicados. Como se pode ver na Figura 6, numa rede com replicação de barramentos, uma falha num barramento tem consequências diferentes conforme o tipo de nodo. Considerando o caso demonstrado na Figura 6, caso o barramento 1 falhe, o nodo 1 vai funcionar com tripla replicação, o nodo 5 vai deixar de ter replicação e os nodos 2 e 3 ficam isolados da rede perdendo a capacidade de comunicar.

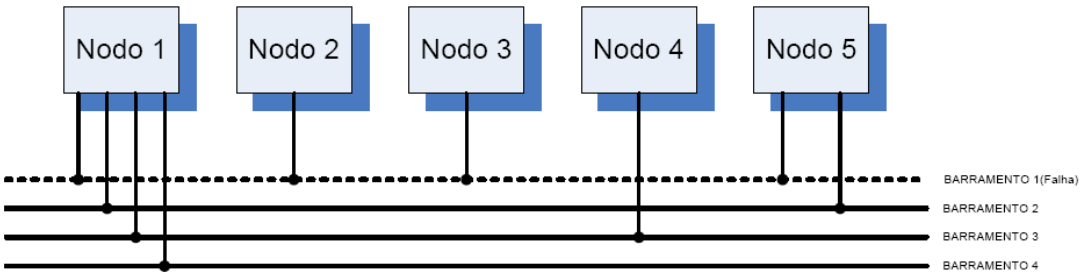


Figura 6: Rede com replicação de barramentos - adaptado de (Silva, et al., 2006).

Se a replicação for gerida ao nível da rede permite esconder todos os detalhes da gestão da replicação de quem desenvolve a aplicação. Além de facilitar o desenvolvimento de aplicações também providencia melhores propriedades de tolerância a falhas, pois a gestão de replicação é desenvolvida, acedida e validada por especialistas e usada por quem desenvolve as aplicações que muitas das vezes não está familiarizado com questões de tolerância a falhas.

Na Figura 7 está representada a arquitectura proposta em (Silva, et al., 2006), designada por DTM - *Dynamic Topology Management*. Com esta arquitectura é possível montar uma rede que preserva todas as características do

CAN. Possui barramentos replicados, aumentando assim a largura de banda e permite a sua reconfiguração em caso de falha. O aumento da largura de banda é feito através da atribuição de barramentos dedicados à comunicação entre dois ou mais nodos, ficando ainda assim, barramentos livres para a comunicação entre os restantes nodos. Para gerir estas tarefas a arquitectura proposta possui dois tipos de componentes, uma NSU - *Network Switch Unit* e duas TMU - *Topology Manager Unit*. A TMU é responsável por definir a topologia usada em cada instante, fazendo isto através do envio de mensagens para a NSU indicando qual o barramento que o nodo deve usar para comunicar. Neste mecanismo os barramentos utilizados numa comunicação entre nodos podem ser comutados durante a transmissão caso seja detectada uma falha num deles. Além disso, esta operação é transparente para o nodo. Como se pode ver ainda na Figura 7, existem duas TMU sendo uma, réplica da outra. Esta replicação é necessária pois assim deixaremos de ter apenas um ponto de falha, começando uma TMU a funcionar em caso de avaria da outra. Esta replicação requer um protocolo para reforçar o determinismo e sincronismo em caso de falha na TMU activa.

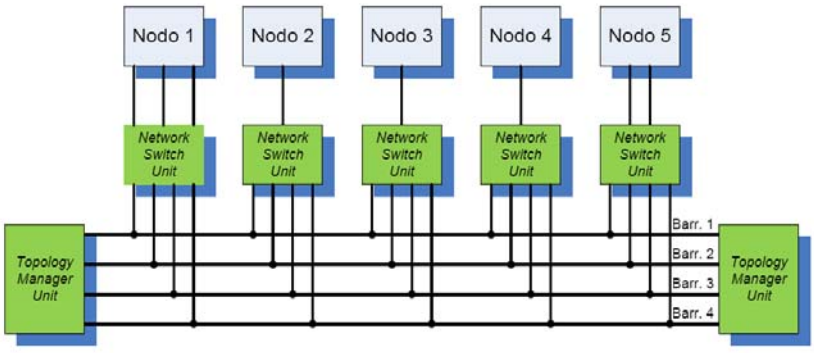


Figura 7: Arquitectura de rede proposta - adaptado de (Silva, et al., 2006).

Com esta arquitectura é possível reagir a falhas permanentes no barramento e reconfigurar dinamicamente a topologia do sistema. Através da substituição de um barramento em falha é possível manter o sistema a funcionar correctamente, possivelmente com uma qualidade de serviço degradada. De notar, que numa situação limite a arquitectura de rede proposta pode ser

configurada como uma estrela virtual, sendo o nodo central o *hub*. Para tal é necessário que este nodo tenha tantas interfaces quanto o número de nodos com o qual vai comunicar.

1.3. Objectivos

O trabalho a realizar tem como principal objectivo a concepção, desenvolvimento e validação dos dois componentes da arquitectura DTM que permitem a gestão dinâmica de uma rede CAN com redundância, nomeadamente a NSU e a TMU. Este objectivo pode ser subdividido nas seguintes partes:

- Conceber e implementar uma TMU - *Topology Manager Unit*. Este componente permite, entre outras funcionalidades, detectar falhas em barramentos redundantes e programar as NSUs para ligarem a interface do respectivo módulo a determinados barramentos. Este componente será implementado em lógica programável, numa placa de desenvolvimento que possui uma FPGA da Xilinx.

- Construir e ensaiar uma NSU - *Network Switch Unit*. As NSUs recebem comandos provenientes da TMU e comutam as ligações consoante as instruções recebidas. Estes componentes também serão implementados em lógica programável.

- Desenvolver novas partes de hardware e adaptar um controlador CAN, o CLAN (Arnaldo S. R. Oliveira), já desenvolvido no DETI.

- Adaptar o controlador do CLAN, alterando algumas das suas características, para facilitar a sua interface com processadores de 8bits.

1.4. Organização da Dissertação

Além deste capítulo introdutório, esta dissertação é composta pelos seguintes capítulos:

- **Capítulo 2 – Sistemas Distribuídos –** Neste capítulo é feito um resumo de conceitos fundamentais sobre sistemas distribuídos. É introduzido o modelo OSI e apresentados os barramentos de campo como a infra-estrutura de comunicação normalmente utilizados em sistemas distribuídos de controlo. Finalmente é feita uma introdução aos sistemas de segurança crítica e de algumas técnicas de tolerância a falhas.

- **Capítulo 3 – Protocolo CAN –** Neste capítulo é feito um resumo das várias partes constituintes da camada lógica e da camada física do protocolo CAN.

- **Capítulo 4 – Tolerância a falhas em sistemas distribuídos de segurança crítica –** Neste capítulo é feito um resumo sobre o estado da arte com a apresentação de algumas abordagens, arquitecturas e protocolos pensados para a sua utilização em sistemas de segurança crítica. Algumas dessas abordagens, arquitecturas e protocolos foram pensados para melhorar a camada lógica e física do CAN do ponto de vista de tolerância a falhas, tornando-o mais adequado para aplicações em sistemas de segurança crítica.

- **Capítulo 5 – Gestão Dinâmica da Topologia em CAN –** Neste capítulo é feita uma descrição detalhada de todas as partes constituintes do sistema Gestão Dinâmica da Topologia em CAN e seus modos de operação. Este é o sistema utilizado como base deste trabalho.

- **Capítulo 6 – Implementação da NSU –** Neste capítulo é apresentada a arquitectura e descrito o desenvolvimento da NSU, como um dos módulos

constituintes do sistema apresentado no capítulo 5. É feita também uma apresentação das ferramentas utilizadas para este efeito.

- **Capítulo 7 – Implementação da TMU –** Neste capítulo é apresentado todo o hardware utilizado para implementar a TMU, um dos módulos constituintes do sistema DTM apresentado no capítulo 5.

- **Capítulo 8 – Conclusões –** Neste capítulo é resumido o trabalho apresentado nesta dissertação, bem como analisados os objectivos alcançados. São também apresentadas algumas possíveis direcções de trabalho futuro.

Os apêndices A e B possuem informação adicional sobre a implementação dos componentes NSU e TMU, partes constituintes do sistema apresentado no capítulo 5.

- **Apêndice A – Descrição detalhada da Implementação da NSU –** Neste apêndice é feita uma descrição adicional de algum hardware utilizado e projectado para a implementação da NSU.

- **Apêndice B – Descrição detalhada da Implementação da TMU –** Neste apêndice é feita uma descrição adicional de algum *hardware* utilizado e projectado para a implementação da TMU.

- **Apêndice C – Programa de monitorização PCAN –** Neste apêndice é feita uma apresentação da ferramenta utilizadas para monitorização de barramentos CAN, o *PCAN-View*. Um pequeno exemplo demonstra como é feito o envio de mensagens.

- **Apêndice D – Lista de Acrónimos –** Este apêndice possui a lista de acrónimos utilizados na dissertação.

Capítulo 2

2. Sistemas Distribuídos

2.1. Conceitos Básicos

Num sistema distribuído os nodos, (sensores, controladores, actuadores, consola) estão fisicamente distribuídos por uma dada área. Para interligar estes elementos é necessária uma rede de comunicação, como por exemplo, um barramento de campo. No caso de o sistema de comunicação ser partilhado simplifica a cablagem e aumenta a flexibilidade do sistema. A troca de mensagens entre os vários elementos é, portanto, efectuada no mesmo sistema de comunicação. Na Figura 8 está representado um exemplo de uma abordagem distribuída para implementar um sistema de controlo baseado num barramento de campo.

Figura 8: Abordagem distribuída para o caso da topologia barramento.

Um nodo pode ser composto por um microprocessador, ou microcontrolador e um controlador de comunicações. O SLIO - *Serial Linked I/O P82C150* da *Philips* é um exemplo de um nodo que contém um microprocessador que prepara os dados a serem transmitidos para o barramento CAN e processa os dados recebidos do barramento CAN. Possui também um controlador CAN que cuida de todas as questões relacionadas com o protocolo CAN, nomeadamente a camada física e ligação de dados do modelo OSI (ver Figura 9). Um nodo pode ser

um componente inteligente e a sua utilização pode trazer alguns benefícios, tais como, a sua facilidade de configuração, teste e auto-verificação. Para além disso, pode acarretar menores custos de produção, maior facilidade de instalação e manutenção e também uma maior flexibilidade.

Existem algumas vantagens e desvantagens na abordagem distribuída relativamente a uma abordagem centralizada (Kopetz, 1997). Como vantagens temos as seguintes características:

- A extensibilidade do sistema que permite adicionar facilmente novos módulos ao sistema. Esta característica permite uma fácil evolução de um sistema, sempre que seja necessário adicionar-lhe novas propriedades ou funções;
- A composabilidade relativamente às propriedades dos nodos. É possível construir sistemas de dimensões consideráveis sem que haja incompatibilidades entre as propriedades específicas de cada nodo;
- O confinamento de erros aos vários nodos. É possível particionar a rede de comunicação de forma a criar zonas de confinamento de erros, sendo por isso mais fácil detectar a origem destes erros e corrigi-los antes que estes afectem o sistema. Numa abordagem centralizada é mais difícil definir uma zona de confinamento de erros porque muitos dos recursos do sistema estão multiplexados por vários serviços.
- A tolerância a falhas através da replicação de nodos ou meios de comunicação.

Como desvantagens temos as seguintes características:

- Criticalidade do sistema de comunicação, que se deve aos erros de transmissão e consequente perda de mensagens, aos conflitos de acesso ao meio de comunicação e dessincronização entre os vários nodos;

- A separação física impede que um nó tenha conhecimento do estado global do sistema num dado instante. Esta situação pode dar origem a atrasos indeterminados na comunicação e falhas.

2.2. Modelo OSI

O modelo OSI – *Open Systems Interconnect* (ISO, 1947), foi formalmente definido pela organização ISO e consiste numa forma comum de conectar computadores. Esta arquitectura é um modelo que divide as redes de computadores em sete camadas, de forma a se obter camadas de abstracção. As várias camadas do modelo OSI estão representadas na Figura 9.

Figura 9: Camadas do modelo OSI.

Cada uma destas camadas presta um conjunto de serviços, baseada num conjunto de protocolos. Quando uma aplicação pretende enviar dados para uma outra aplicação, esses dados são entregues à camada *Aplicação*, camada 7, através dos serviços de comunicação existentes. Cada camada invoca os serviços da camada inferior fornecendo-lhe os dados com alguma informação de controlo adicionada. A camada física é responsável pelo envio dos dados. O sistema que

recebe os dados fá-lo por intermédio da camada física. Posteriormente, cada camada retira a informação de controlo respectiva e passa a restante à camada superior, até à camada *Aplicação*.

2.3. Barramentos de Campo

Designa-se por barramento de campo o sistema de comunicação que interliga os equipamentos ou nodos de um sistema de controlo distribuído, (e.g., sensores, actuadores, controladores, reguladores). Actualmente o *ProfiBus* (Profinet, 1989), o *WorldFIP* (WorldFip, 1993), o *P-Net* (IPUO, 1983), o *DeviceNet* (ODVA, 2008) e o CAN, são alguns dos barramentos de campo mais utilizados.

Em barramentos de campo as mensagens transferidas entre nodos contêm geralmente pequenas quantidades de informação organizada em bytes. O comportamento temporal é bem definido, ou seja, é possível determinar os atrasos de comunicação no pior caso. É necessário um esforço adicional de computação e de largura de banda para efectuar as várias operações entre as várias camadas do modelo OSI. Este esforço introduz atrasos inerentes ao processamento dos serviços em todas as camadas do modelo de referência, bem como da largura de banda com informação de controlo. Para reduzir este atraso os barramentos de campo costumam usar uma versão adaptada do modelo OSI (ver Figura 10).

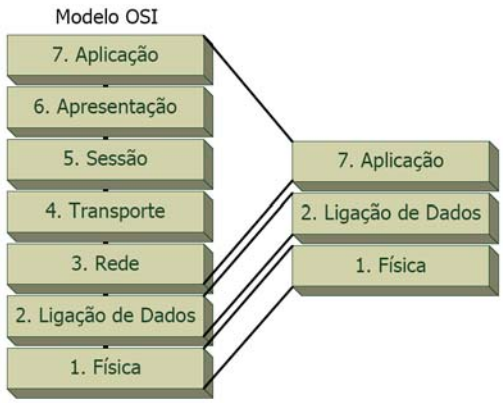


Figura 10: Versão colapsada do modelo OSI aplicado a barramentos de campo.

Existem várias topologias utilizadas em barramentos de campo, sendo as mais conhecidas a topologia em barramento, a topologia em anel, o emaranhado, a árvore e a estrela. Um pequeno esquema representativo de cada tipo de topologia pode ser visto na figura seguinte (ver Figura 11).

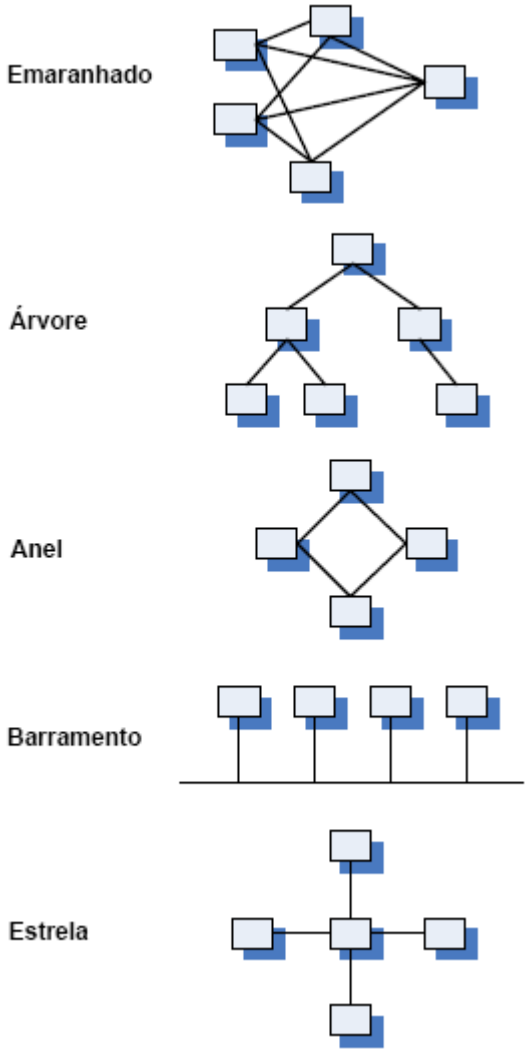


Figura 11: Exemplos de topologias genéricas.

Dependendo do tipo de aplicação a que cada uma destas topologias se destina pode haver vantagens e desvantagens ao nível das ligações, ao nível da

cablagem e da tolerância a falhas. Na Tabela 1, estão apresentadas algumas vantagens e desvantagens de cada uma destas topologias de ligação.

Topologia	A favor	Contra
Emaranhado	Ligações ponto-a-ponto. Vários percursos alternativos.	Necessidade de encaminhamento (*routing*). Interligações complexas e de difícil manutenção.
Árvore	Ligações ponto-a-ponto. Comunicação simultânea em ramos paralelos. Isolamento entre ramos.	Atrasos na comunicação entre nodos em ramos distantes – necessidade de encaminhamento.
Anel	Cablagem simplificada. Maior robustez em relação a erros de transmissão – facilidade de confirmação.	Ligações ponto-a-ponto mas usadas como meio partilhado - controlo do acesso ao meio mais complexo.
Barramento	Cablagem simplificada. Nodos comunicam directamente, não é necessário encaminhamento.	Meio de comunicação partilhado - controlo do acesso ao meio mais complexo.
Estrela	Maior tolerância a falhas devido a não existir um meio partilhado. Ligações ponto-a-ponto. Isolamento entre ramos.	Necessita de um *hub* central.

Tabela 1: Camada física - Topologia de interligação - adaptado de (Almeida, 2001).

Ao nível do meio de comunicação, as várias tecnologias utilizadas podem ter vantagens ou desvantagens dependendo da aplicação a que se destina. Na tabela seguinte, Tabela 2, estão apresentadas algumas vantagens e desvantagens das várias tecnologias utilizadas no meio de comunicação, camada física.

Tipo de meio		Vantagens	Desvantagens
Cablagem de cobre		Interfaces e cabos baratos.	Sensível a interferências electromagnéticas (EMI).
Fibra óptica		Imune a EMI, baixa atenuação, grande largura de banda.	Interfaces e cabos caros.
Sem fios	**Infra-vermelhos**	Mobilidade.	Nodos em "linha de vista", sensível a interferências.
	Rádio frequência		Sensível a interferências.

Tabela 2: Camada física - Meio.

Em termos de ligação de dados, o acesso ao meio pode ser controlado ou não. No caso de o acesso ser controlado, existe uma forma acordada de controlo para acesso ao meio que determina quando é que cada nodo pode transmitir, por exemplo, mestre-escravo (*master-slave*), testemunho (*token*). No caso do acesso não controlado, cada nodo inicia a transmissão assim que a respectiva aplicação assim o determina. Existem métodos que resolvem as colisões que ocorrem, por exemplo, o CSMA-CD – *Carrier Sense Multiple Access with Collision Detection* ou o CSMA-DCR – *Carrier Sense Multiple Access with Deterministic Collision Resolution*. Para fazer o escalonamento do tráfego é utilizada a arbitragem como método para resolver conflitos no acesso ao meio. Como exemplos de sistemas de comunicação onde são utilizados métodos de arbitragem temos, o WorldFIP (WorldFip, 1993) onde a arbitragem é centralizada e o escalonamento qualquer, o CAN onde a arbitragem é distribuída (CSMA-DCR) com escalonamento por prioridades fixas.

2.4. Segurança Crítica

Os sistemas de segurança crítica são assim designados por várias razões, de entre as quais, o consequente prejuízo elevado em caso de falha, apertadas restrições sobre a utilização de recursos e os ambientes onde operam que são perigosos. Como consequência de tais factores, estes sistemas são obrigados a operar numa margem muito estreita de erro. Para além disso, o tempo disponível para tomar acções correctas no caso de haver uma falha em sistemas tempo-real pode ser extremamente limitado, especialmente nas fases mais críticas. Uma falha grave num sistema deste tipo pode resultar na morte ou ferimentos graves em

pessoas, estragos ou perda completa de equipamento ou até mesmo numa tragédia ambiental. Quando do bom funcionamento do sistema dependem vidas humanas, a probabilidade de falha para o qual o sistema é projectado pode chegar a 10^{-10} por cada hora de funcionamento, (Nissanke, cop. 1997 p. 323).

O desenvolvimento de sistemas de segurança crítica começa com o reconhecimento de que a segurança é um objectivo muito importante a ser considerado ao nível do sistema, que não deve ser desvalorizado a favor de outros objectivos do projecto e poupança na tentativa de redução do custo final. Assim, um sistema de segurança crítica deve ter os seguintes atributos: disponibilidade, fiabilidade, segurança, confidencialidade e integridade. Segundo (Nissanke, cop. 1997 p. 311), o desenvolvimento de sistemas de segurança crítica começa também por definir o que é a segurança crítica, isto envolve:

a) Identificar os riscos, (perigos), envolvidos.
b) O que é que a segurança significa em termos de comportamentos indesejados do sistema. Estes devem ser explicitamente identificados nas especificações pretendidas e são referidas como requerimentos de segurança.
c) Com que rigor devem ser observados os requisitos de segurança. Estes requisitos devem ser expressos através de critérios probabilísticos reflectindo parcialmente os vários níveis de risco.
d) O que a segurança significa em termos de quantificação das perdas em caso de falha.

Os requisitos c) e d) juntos constituem o que normalmente se chama de "risco". Assim, grandes provisões de riscos reflectem em custos elevados devido a uma elevada antecipação da probabilidade de falha.

O objectivo das medidas que se tomam no que respeita a tolerância a falhas, no contexto de sistemas de segurança crítica, é localizar os efeitos de falhas para que o desempenho de todo o sistema não seja afectado por falhas em pequenos componentes e que o sistema falhe em segurança em caso de falha em

componentes de maior importância. A complexidade de tais mecanismos em função do grau de tolerância a falhas, criticalidade do sistema, definem o rácio custo-eficácia na altura do projecto e desenvolvimento de tais sistemas.

2.5. Tolerância a Falhas

Tolerância a falhas é uma propriedade que permite que um sistema continue a funcionar normalmente, ou num modo degradado, mesmo após falhas em alguns dos seus componentes. Esta propriedade está associada a sistemas de segurança crítica e sistemas de alta disponibilidade. É possível aplicar as técnicas de tolerância a falhas ao nível da arquitectura (tolerância a falhas sistemática), e ao nível da aplicação (tolerância a falhas especifica da aplicação). A tolerância a falhas sistemática permite diminuir a complexidade da aplicação, transferindo-a, para hardware e software adicionais.

Um nodo de um sistema distribuído pode ser considerado um ponto de falha do sistema. Para dar a volta a este problema é possível utilizar a redundância ao nível do nodo, através da replicação deste elemento. Existem várias técnicas relacionadas com tolerância a falhas em sistemas distribuídos baseados em nodos como pontos de falha. O controlo de presença é uma técnica que permite saber quais os nodos operacionais, utilizando para isso uma *flag* num registo que é activada periodicamente. A gestão da redundância é uma outra técnica que efectua a detecção de discrepâncias entre as saídas de nodos replicados e controla a entrada e saída de funcionamento das réplicas.

Quando se incorporam técnicas de tolerância a falhas ao nível do nodo é necessário utilizar unidades tolerantes a falhas (*FTUs – Fault-Tolerant Units*). Estas unidades funcionam da mesma forma que o nodo base, mas possuem serviços suplementares que lhes permitem ter um certo nível de tolerância a falhas.

Existem vários tipos de falhas com que as FTU têm de lidar, nomeadamente:

- Falhas do tipo falha-silêncio;
- Falhas do tipo falha-pára ou falha-consistente;
- Falhas do tipo falha-inconsistente (maliciosa ou bizantina).

De seguida será feita a descrição de cada um destes tipos de falha.

2.5.1. Falhas do Tipo Falha-Silêncio

Falhas do tipo falha-silêncio, são falhas que quando ocorrem, o nodo ou fornece respostas correctas ou então simplesmente não responde. Quando o nodo base de uma FTU só apresenta falhas do tipo falha-silêncio, é necessário no mínimo a utilização de dois nodos (ver Figura 12). No caso de um dos nodos falhar o outro continua a responder.

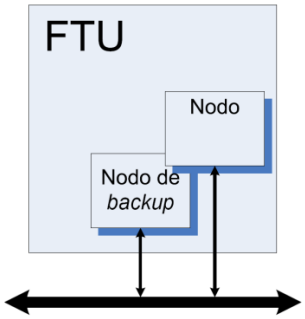


Figura 12: FTU tolerante à falha do tipo falha-silêncio com dois nodos - adaptado de (Almeida, 2001).

2.5.2. Falhas do Tipo Falha-Pára ou Falha-Consistente

Falhas do tipo falha-pára ou falha-consistente, são falhas onde o nodo, após a ocorrência de uma falha, pode continuar a enviar um valor errado mas

consistente. Neste caso são necessários três nodos na FTU e um mecanismo para determinar de entre as três saídas a que está correcta, (*TMR – Triple Modular Redundancy*) (ver Figura 13).

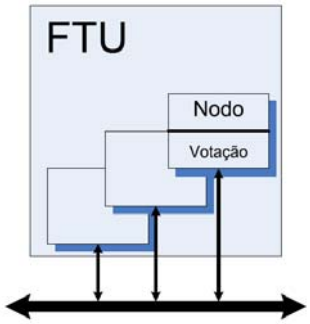


Figura 13: FTU com três nodos e mecanismo de votação TMR - adaptado de (Almeida, 2001).

2.5.3. Falhas do Tipo Falha-Inconsistente (maliciosa ou bizantina)

Falhas do tipo falha-inconsistente (maliciosa ou bizantina), são falhas em que o nodo, após uma falha, pode continuar a produzir valores errados e inconsistentes. Neste caso é necessário utilizar quatro nodos e um protocolo que permita determinar qual o nodo que está a produzir um valor errado (ver Figura 14). Com este protocolo todos os nodos da FTU trocam mensagens entre si com o valor das respectivas saídas, (*Byzantine-Resilient Agreement Protocol*).

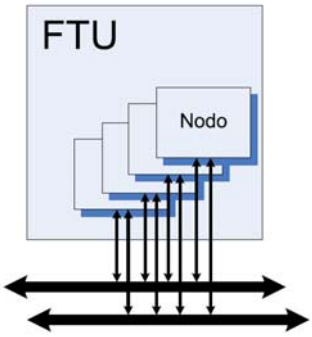


Figura 14: FTU com quatro nodos e protocolo (*Byzantine-Resilient Agreement Protocol*) – adaptado de (Almeida, 2001).

Capítulo 3

3. O Protocolo CAN

3.1. Introdução

O protocolo CAN - *Controller Area Network* (CiA, 2001-2008), conta já com 22 anos de existência, foi apresentado pela primeira vez no congresso *"Society of Automotive Enginners (SAE)"* por *Robert Bosch GmbH*. Não será com certeza um exagero dizer que é hoje em dia um dos mais bem sucedidos protocolos de barramentos de campo. Foi inicialmente desenvolvido para a indústria automóvel para suportar comunicações de baixo nível entre módulos, (por exemplo, nos sistemas de travagem como o ABS - *Anti-lock Braking System,* ou nos sistemas de estabilidade como o ESC - *Electronic Stability Control*, mas hoje em dia é também usado noutros tipos de veículos tais como, comboios, aviões, navios, entre outros. O sucesso da utilização do CAN deve-se ao baixo preço dos controladores que são produzidos em grande quantidade e às suas características técnicas que o tornam relativamente fiável. A utilização do CAN em aplicações de segurança crítica não pode ser feita utilizando-o na sua forma nativa. Por exemplo em sistemas de segurança critica como o *x-by-wire,* normalmente utilizados nos automóveis, é necessário utilizar redundância e características que permitem ao sistema falhar de uma forma segura (Touloupis, et al., 2003).

O CAN é baseado na topologia barramento e utiliza o mecanismo de acesso ao meio CSMA/DCR para resolver as colisões. A transmissão de dados no CAN é feita através de um modelo binário de bits "dominantes" e "recessivos" onde o dominante representa o estado lógico '0' e o recessivo representa o estado lógico '1'. O controlo do acesso ao meio por parte dos nodos é feito utilizando um sistema de arbitragem que utiliza este modelo binário de bits "dominantes" e "recessivos". Durante o processo de arbitragem, o nodo que pretende aceder ao

barramento para transmitir, "escuta" o barramento e compara o valor do bit que está a tentar transmitir com o valor presente no barramento. Se no barramento estiver presente um valor dominante, isto significa que outro nodo está a transmitir uma mensagem de maior prioridade e o nodo que estava a tentar aceder ao barramento perde a arbitragem. Para evitar a omissão ou perda da mensagem, o nodo atrasa a transmissão até haver outra arbitragem.

O protocolo CAN define duas camadas do modelo OSI, a camada dois, ligação de dados, e parte da camada um, camada física (ver Figura 15).

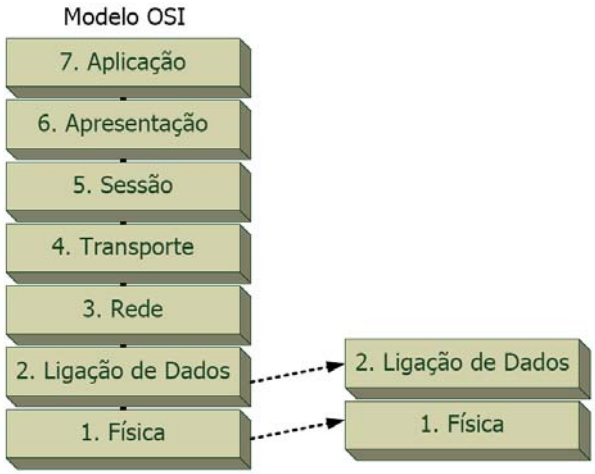


Figura 15: Camadas do modelo OSI utilizadas pelo CAN.

Existem especificações standard definidas pela ISO para o CAN, como por exemplo a ISO 11898-1:2003 que especifica a camada de dados e as sinalizações físicas para um protocolo de comunicações série que suporta controlo tempo-real e multiplexagem para uso em veículos (ISO, 2003).

A troca de informação entre nodos CAN é feita através de tramas que possuem um identificador que pode ter 11 bits num formato base e 29 bits num formato extendido. O campo de dados das tramas é de tamanho variável e pode variar de 0 até 8 bytes (ver Figura 17). O identificador da trama define a prioridade da mensagem durante o processo de arbitragem e informa os restantes nodos da rede sobre o conteúdo desta mensagem. As tramas não possuem informação sobre o destinatário, sendo enviadas para todos os nodos, logo cada nodo tem que decidir quais as tramas que lhes interessam. Desta forma os nodos

recebem as tramas em simultâneo e podem tomar acções sobre a mesma mensagem. O identificador de uma trama é produzido apenas por um nodo na rede, sendo portanto original e específico do nodo em questão.

Muitos dos problemas de fiabilidade do CAN estão directamente relacionados com a sua topologia de rede. O principal problema provém do facto de existir apenas um barramento, que representa um único ponto de falha, onde estão electricamente ligados vários nodos entre si sem que haja um adequado confinamento de erros. Um nodo que, por algum motivo, comece a gerar erros vai comprometer as comunicações até que o contador de erros exceda um determinado valor. Na Figura 47, na página 68, no caso A está representado um exemplo deste tipo de falha onde, por exemplo, um nodo transmite para o barramento um valor de bit fixo bloqueando assim as comunicações entre os restantes nodos. Outra situação que pode ocorrer está representada na Figura 47, caso B, onde um possível corte no meio de comunicação divide o barramento, criando assim uma partição, isolando os nodos um e dois dos nodos três e quatro. O último caso, caso C, representa uma falha no meio de comunicação entre o nodo e o barramento. Este tipo de falha isola o nodo da restante rede mas não faz com que as comunicações entre os restantes nodos terminem. Este tipo de falha é a que menos impacto tem nas comunicações.

3.2. Camada Lógica

Nos subcapítulos seguintes será descrito o princípio da arbitragem e o formato das tramas no CAN.

3.2.1. Arbitragem no Acesso ao Barramento

Uma vez que no CAN o barramento é partilhado, é necessário haver um mecanismo de controlo de acesso ao meio que evite colisões. O CAN utiliza o CSMA-DCR - *Carrier Sense Multiple Access with Deterministic Collision Resolution*. Quando vários nodos começam a transmitir simultaneamente, apenas o nodo que estiver a transmitir a mensagem com maior prioridade ganha o acesso ao barramento. Este processo de arbitragem é não destrutivo, ou seja, nenhuma trama é destruída, não havendo por isso necessidade de retransmissões.

Na fase de arbitragem (ver Figura 16) os nodos concorrentes transmitem cada um o identificador da sua mensagem bit a bit, verificando ao mesmo tempo se o nível presente no barramento corresponde ao nível que está a ser transmitido. Quando um nodo encontra um nível no barramento diferente ao que está a ser transmitido, então ele perde o acesso ao barramento porque outro nodo está a transmitir uma mensagem de maior prioridade. Todos os nodos que perdem o acesso, tornam-se automaticamente receptores da mensagem com a maior prioridade.

Existem dois formatos para as mensagens CAN, o chamado *"base format"* que possui 11bits no campo do identificador e o *"extended format"* com 29 bits no identificador. No caso de dois nodos concorrerem ao acesso ao barramento para transmitir uma mensagem, uma com identificador de 11bits e outra com identificador de 29bits, mas ambas com identificador de base iguais, ganha sempre o acesso ao barramento o nodo que pretende enviar a mensagem com identificador de 11bits. É esta a forma que o protocolo tem de resolver colisões entre mensagens com identificadores base iguais.

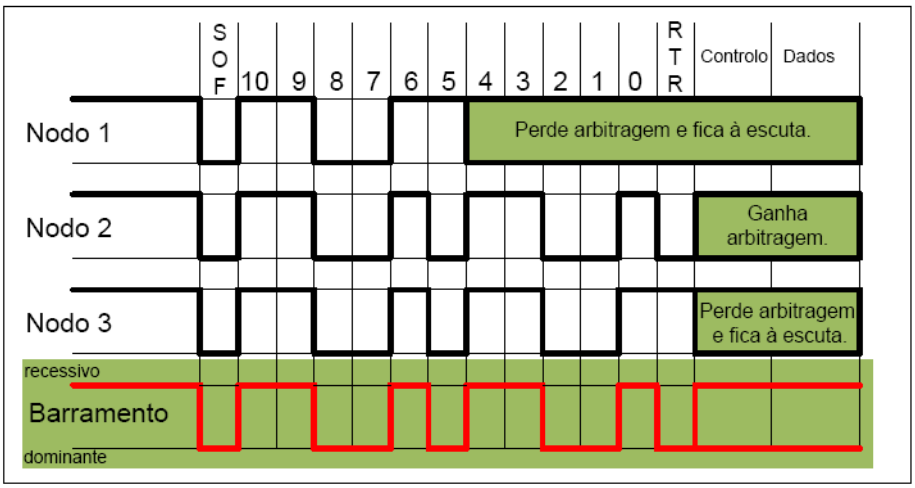


Figura 16: Exemplo de um processo de arbitragem de acordo com o protocolo CAN - adaptado de (CiA, 2001-2008).

3.2.2. Tramas CAN

O protocolo CAN define os seguintes quatro tipos de trama:

- Trama de dados (*Data frame*)
- Trama pedido de transmissão (*Remote frame*)
- Trama de sinalização de erro (*Error frame*)
- Trama de sinalização de sobrecarga (*Overload frame*)

De seguida será feita a descrição de cada um dos tipos de trama.

3.2.2.1. Trama de Dados

Na Figura 17 está representada uma trama de dados que consiste nos seguintes campos: SOF - *Start-of-Frame*, Arbitragem (*Arbitration Field*), Controlo (*Control Field*), Dados (*Data Field*), CRC - *Cyclic Redundant Check*, ACK - *Acknowledgement* e EOF - *End-of-Frame*. O campo de dados é opcional podendo ter entre 0 e 8 bytes. O campo EOF - *End-Of-Frame Flag* sinaliza o final da trama de dados e é constituído por uma sequência de sete bits no estado recessivo.

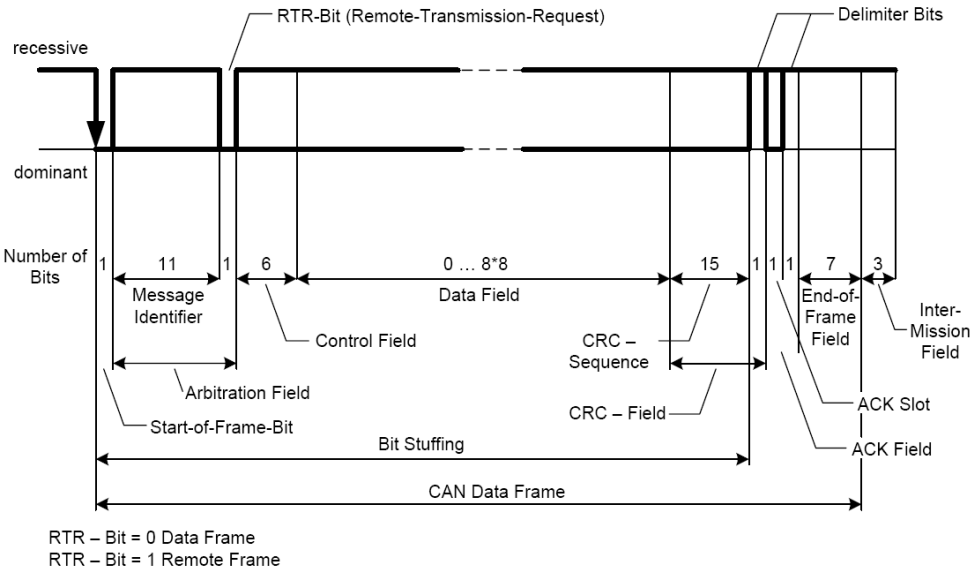


Figura 17: Formato da trama de dados e trama remoto (formato base) - adaptado de (Etshberger, 2001).

Bit de inicio de trama (*Start-of-Frame Bit*):

Este bit marca o início de uma trama de dados ou trama de pedido de transmissão (*Remote Frame*), sendo representado por apenas um bit dominante. Só é permitido a um nodo iniciar a arbitragem quando o barramento está livre.

Campo arbitragem (*Arbitration Field*):

Na Figura 18, está representado o campo de arbitragem numa trama CAN. Este campo é composto pelo identificador e o bit RTR. No formato base, o identificador possui apenas 11bits, o que permite 2048(2^{11}) tramas diferentes. As tramas de dados e as tramas de pedido de transmissão são diferenciadas pelo bit RTR. Para que as tramas de dados sejam mais prioritárias que as tramas de pedido de transmissão, o bit RTR de uma trama de dados é transmitido no estado

dominante (valor lógico 0), enquanto numa trama de pedido de transmissão o bit RTR é transmitido no estado recessivo (valor lógico 1).

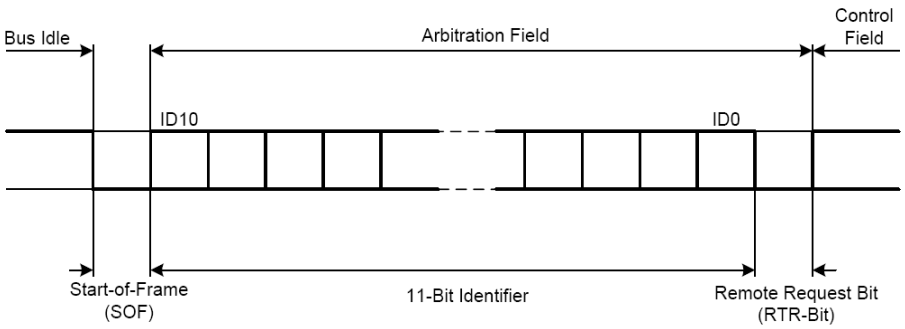


Figura 18: Formato do campo arbitragem - adaptado de (Etshberger, 2001).

Campo de controlo (*Control Field*):

O campo de controlo (ver Figura 19) possui seis bits incluindo o DLC, de acordo com as especificações do CAN 2.0 B. O primeiro bit deste campo, *identifier extension bit* (IDE), permite fazer a distinção entre os formatos *"base"* ou *"extended"*. No formato base o IDE Bit é transmitido no estado dominante. O segundo bit, r0, está reservado para futuras expansões do CAN e é transmitido no estado dominante. Os últimos quatro bits deste campo, conhecidos como *data lenght code* (DLC), permite definir quantos bytes de dados estão a ser transmitidos.

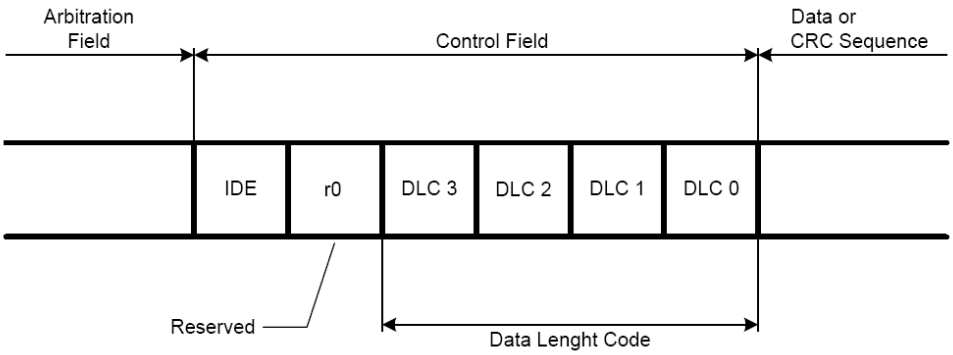


Figura 19: Formato do campo de controlo - adaptado de (Etshberger, 2001).

Campo de dados (*Data Field*):

Este campo contém os dados a transmitir numa trama CAN. Os dados podem ir desde 0 a 8 bytes, sendo os bits mais significativos os primeiros a ser transferidos.

Campo CRC (*CRC Field*):

Este campo permite ao nodo receptor verificar se os dados estão corrompidos ou não. Com este método é possível garantir uma grande probabilidade na detecção de erros. O campo CRC consiste numa sequência de 15bits mais um bit CRC que serve como delimitador (ver Figura 20).

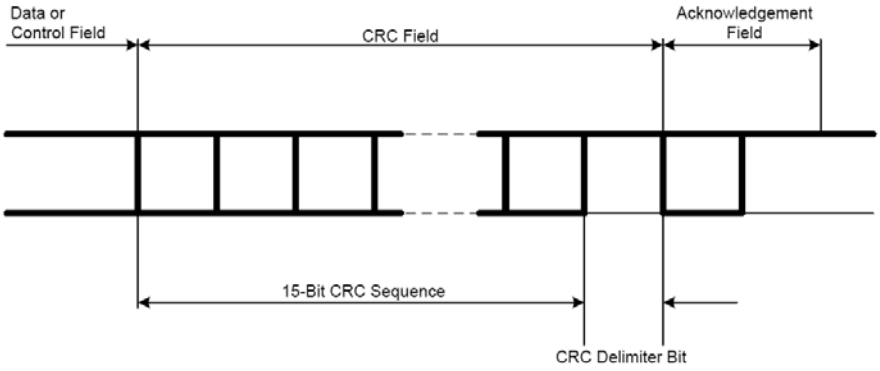


Figura 20: Formato do campo CRC - adaptado de (Etshberger, 2001).

Campo de reconhecimento (*Acknowledgement Field*):

Este campo é constituído por um ACK–*Slot* - *Acknowledgement Slot* de dois bits de reconhecimento. O primeiro dos bits é um *slot* de reconhecimento e o segundo um delimitador (ver Figura 21).

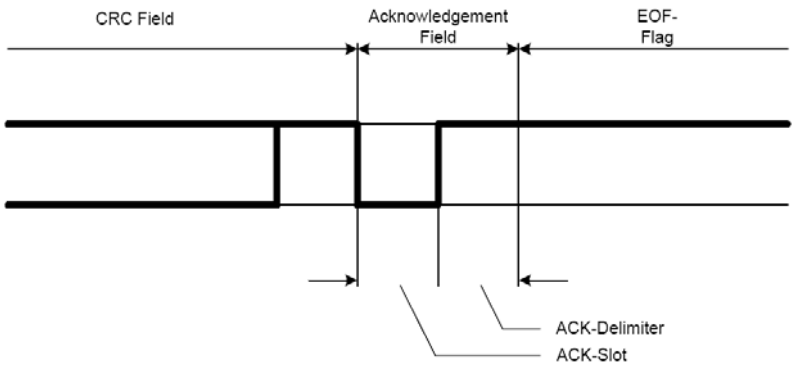


Figura 21: Formato do campo *acknowledgement* - adaptado de (Etshberger, 2001).

Estes dois bits de reconhecimento são transmitidos com alta prioridade por um nodo transmissor, que depois aguarda a confirmação do reconhecimento da trama transmitida por pelo menos um dos nodos receptores. O nodo receptor que receber uma trama correctamente, sem erros, reporta ao transmissor o envio bem sucedido, sobrepondo o bit no estado recessivo enviado pelo transmissor pelo estado dominante durante o *slot* de reconhecimento.

3.2.2.2. Trama Remoto

Com este tipo de trama é possível requisitar dados a um determinado nodo. O bit RTR define se a trama é de dados ou se é uma trama de requisição de dados, ou seja, se o bit RTR da trama estiver a um (RTR-Bit = 1) trata-se de uma trama remoto, se o bit RTR estiver a zero (RTR-Bit = 0) então trata-se de uma trama de dados. Esta diferença no nível do bit RTR permite em caso de colisão de duas tramas com o mesmo identificador na fase de arbitragem, que a trama de dados seja mais prioritária vencendo assim a arbitragem. Para além da diferença do estado do bit RTR, as tramas remoto diferem também das tramas de dados no campo de dados, pois este campo não existe neste caso.

3.2.2.3. Trama de Sinalização de Erro

No protocolo CAN, uma sequência de bits com mais de cinco bits de igual valor, numa trama de dados ou trama remoto, é considerada uma trama de sinalização de erro. Esta sequência viola a regra da técnica *bit stuffing* (técnica descrita na subsecção 3.3.1) obrigando assim o transmissor a repetir a transmissão. Uma trama de sinalização de erro é constituída por dois campos, o campo *"error flag"* e o campo *"error delimiter"* (ver Figura 22). A detecção de erros numa trama de erro ou trama de sobrecarga obriga a uma nova transmissão de uma trama de sinalização de erro.

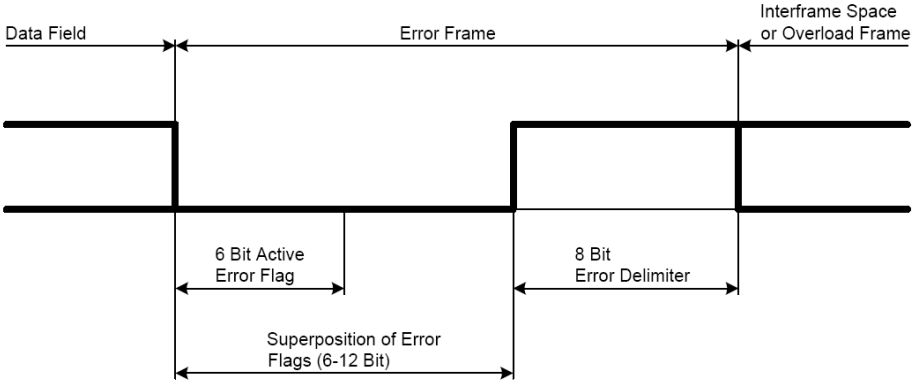


Figura 22: Formato de uma trama de erro activo - adaptado de (Etshberger, 2001).

A *flag* de erro (*Error Flag*) pode ter as duas formas seguintes, *flag* de erro activo ou *flag* de erro passivo. Como se pode ver na Figura 22, a *flag* de erro activo consiste numa sequência de seis bits dominantes. A *flag* de erro passivo consiste numa sequência de seis bits recessivos, caso não tenham sido sobrepostos por bits dominantes de outros nodos.

Após a transmissão de uma *flag* de erro, cada um dos nodos envia bits no estado recessivo e monitora o nível de barramento até detectar um bit recessivo. Quando o bit no estado recessivo é detectado, o nó inicia a transmissão de mais sete bits no estado recessivo. Este campo, *"error delimiter"*, indica se foi o nodo o primeiro a detectar o erro.

3.2.2.4. Trama de Sinalização de Sobrecarga

É composta por um campo *"overload flag"* e um campo *"overload delimiter"*. Este tipo de trama é utilizado por um nodo receptor para pedir um atraso no envio da próxima trama de dados ou trama de pedido de transmissão, ou quando o receptor detecta um bit dominante no ultimo bit do delimitador de fim de trama.

3.3. Camada Física

Nas subsecções seguintes serão descritas algumas características da camada física do CAN, tais como, a codificação dos bits, a temporização dos bits e sincronização, a interdependência entre taxa de transmissão e comprimento do barramento, o meio físico e os parâmetros do meio físico.

3.3.1. Codificação dos Bits

O método utilizado para a codificação dos bits em CAN é o NRZ - *Non-Return-to-Zero* (CiA, 2001-2008). Como se pode ver na Figura 23, com a codificação NRZ o sinal fica constante durante o tempo de bit, ao contrário da codificação *Manchester*, o que pode ser uma vantagem uma vez que aumenta a imunidade ao ruído. O facto de o sinal ficar constante durante o tempo de bit faz com que não haja uma mudança de estado no barramento para cada bit transmitido, o que dificulta a sincronização entre os nodos. No caso de uma sequência de bits iguais (recessivos ou dominantes), os nodos podem ter dificuldades na sincronização uma vez que podem não conseguir reconhecer os limites dos bits. Para ultrapassar este problema é utilizada uma técnica chamada *bit stuffing* que permite uma melhor sincronização das mensagens por parte dos nodos receptores. Quando o transmissor pretende enviar uma sequência de cinco bits de igual valor, introduz na mensagem um bit de valor complementar a seguir

à sequência de 5 bits (ver Figura 24). Como é óbvio o receptor tem como tarefa retirar os *stuff -bits* na mensagem (*destuffing*).

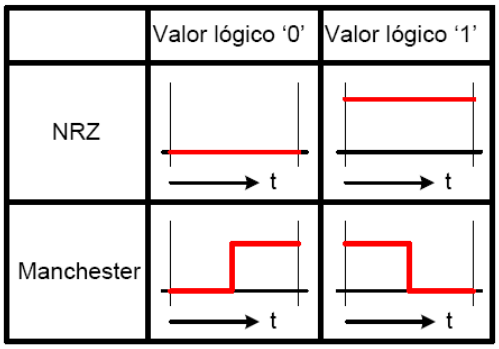


Figura 23: Comparação entre o código NRZ e o código Manchester - adaptado de (CiA, 2001-2008).

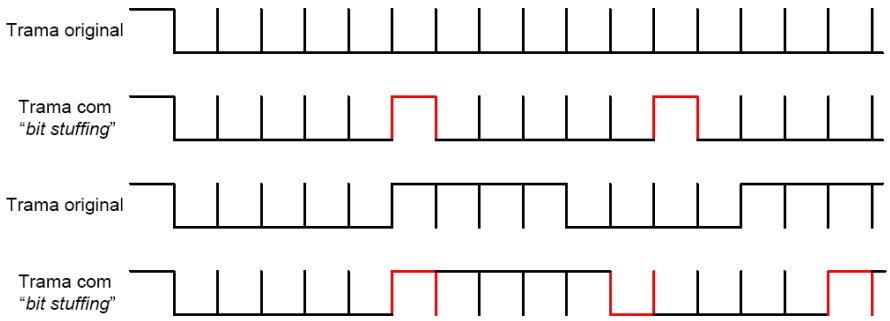


Figura 24: Exemplos da técnica *"bit stuffing"* usada no protocolo CAN.

A codificação NRZ com *bit stuffing* subjacente ao protocolo CAN abrange os segmentos de trama SOF - *Start-of-Frame*, Arbitragem (*Arbitration Field*), Controlo (*Control Field*) e Campo de Dados (*Data Field*) assim como a sequência CRC - *Cyclic Redundancy Check*. O CRC é o mecanismo utilizado para a detecção de erros. Os restantes campos de uma mensagem de dados, nomeadamente, o delimitador de CRC, o campo ACK e o EOF - *End-of-Frame* têm uma forma fixa (estado recessivo) e são transmitidos sem *bit stuffing*. O mesmo se aplica a tramas de erro e de sobrecarga.

3.3.2. Temporização dos Bits e Sincronização

Numa rede CAN os nodos possuem o seu próprio relógio e durante a transmissão de dados nenhum sinal de relógio é enviado. A sincronização é feita dividindo cada bit da trama num número de segmentos: sincronização, atraso de propagação, fase 1 e fase 2. O tamanho de cada segmento pode ser ajustado dentro de determinados limites. Na Figura 25, estão representadas as várias fases num tempo de bit. O tempo de bit é composto por um segmento de sincronização, por um segmento de atraso de propagação e dois segmentos de fase.

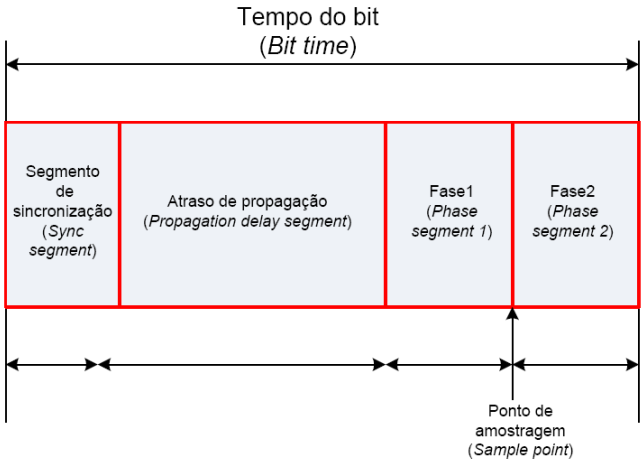


Figura 25: As várias fases num tempo de bit - adaptado de (CiA, 2001-2008).

O protocolo CAN utiliza uma arbitragem *bitwise* não destrutiva de acordo com a sua característica *wired-and* em que o estado dominante se sobrepõe ao estado recessivo. Desta forma, para que não haja problemas durante a arbitragem é necessário garantir que o tempo de propagação de um sinal desde o transmissor até ao receptor e de volta ao transmissor deve ser completado num tempo de bit. O segmento de atraso de propagação leva em consideração a propagação do sinal no barramento assim como o atraso causado pelos nodos transmissor e receptor.

3.3.3. Interdependência entre Taxa de Transmissão e Comprimento do Barramento

O tempo de propagação utilizado para definir o segmento Atraso de propagação (*Propagation delay segment*) é o tempo que um sinal leva para percorrer a distância de um nodo até ao outro (estando estes nas extremidades do barramento), mais a soma do atraso causado pela transmissão e recepção nos nodos, mais o tempo de sincronização e o tempo que o sinal leva a regressar. Um nodo que por exemplo pretende aceder ao barramento para transmitir uma trama, só toma decisões sobre o estado do barramento depois de este tempo ter passado, para garantir que o estado do barramento não tenha sido entretanto alterado por outro nodo.

Na Figura 26 está representada a relação que existe entre o comprimento do barramento e a velocidade de transmissão. Podemos ver que para comprimentos inferiores a quarenta metros o sistema pode operar a 1Mbps, a partir dessa distância, para distâncias superiores, a velocidade de transferência de dados diminui linearmente.

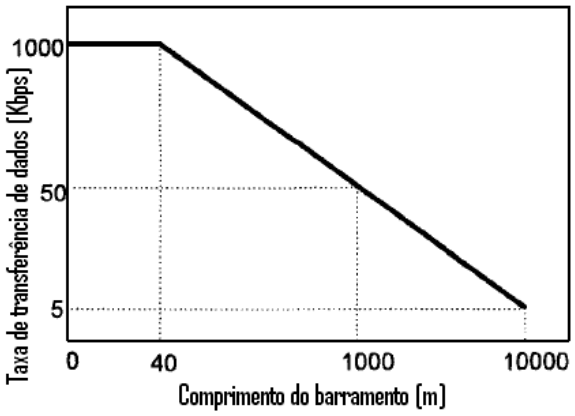


Figura 26: Relação entre a taxa de transferência de dados e o comprimento do barramento - adaptado de (Controller Area Network Implementation in Microwave Systems, 2002).

Para comprimentos de barramento superiores a mil metros devem ser usados controladores de linha especiais.

3.3.4. Meio Físico

A utilização de um meio de transmissão no CAN é possível desde que este permita a representação do estado dominante e recessivo. Como exemplos de meios de transmissão que preenchem este pré requisito temos os meios eléctricos, os ópticos, e os meios sem fios (*wireless*).

Para meios eléctricos as tensões de saída para o barramento são definidas pela ISO 11898-2, ISO11898-3, SAE J2411 e ISO 11992 (ISO, 1947).

Para meios ópticos o modo recessivo é representado por "escuro", (ausência de luz), e o bit dominante por "luz". O meio físico pode ser uma fibra óptica permitindo assim isolamento galvânico e uma grande eficiência. Esta solução pode ser interessante para aplicações a funcionar em ambientes onde existe perigo de explosão ou em ambientes com ruído electromagnético muito forte, (Etshberger, 2001).

O meio de transmissão mais utilizado para implementar redes CAN é o cabo par diferencial entrançado. Para a electrónica em veículos é frequentemente usado apenas um fio para implementar o barramento. Esta técnica permite reduzir a cablagem nos veículos, questão importante devido à falta de espaço e poupança.

Existem também soluções designadas por PLC - *Power Line Comunication* que permitem transmitir sinais CAN na mesma linha da alimentação. O DCAN250 é um componente disponível no mercado que permite este tipo de comunicação (YAMAR, 2007). Este componente funciona como um *transceiver* CAN com capacidade para utilizar a linha de alimentação para a comunicação entre nodos, (ver Figura 27).

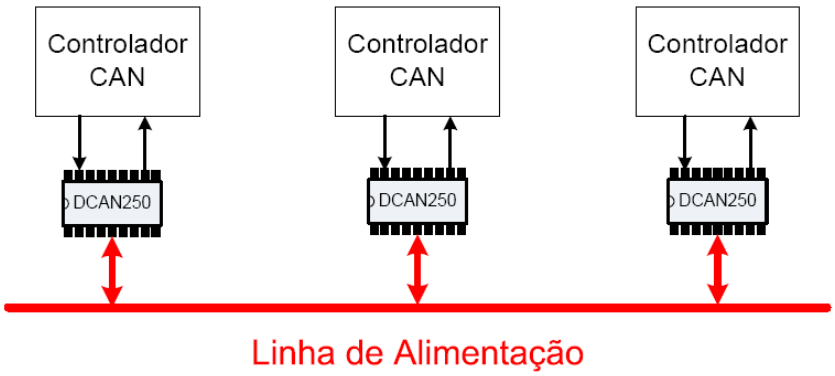


Figura 27: Modo de utilização do DCAN250 - adaptado de (YAMAR, 2007).

3.3.5. Parâmetros do Meio Físico

Os parâmetros dos meios físicos de transmissão tornam-se importantes quando o comprimento do barramento é aumentado. O tempo de propagação do sinal, a resistência e secções transversais do cabo são factores a considerar aquando do dimensionamento da rede. Nos cabos a tensão diminui ao longo do cabo e para grandes comprimentos essa diminuição na tensão deve ser levada em consideração. A secção transversal do cabo necessária é calculada pela queda de tensão permitida do sinal entre os dois nodos mais afastados no sistema e a resistência total de todos os receptores conectados. A queda de tensão permitida tem de ser tal que o nível do sinal possa ser devidamente interpretado pelo nodo receptor.

O máximo comprimento para um barramento é dado pela mínima tensão diferencial num nodo receptor para que um nível de bit dominante seja devidamente reconhecido. Um receptor reconhece um bit dominante se a tensão diferencial estiver acima de 1V. Podemos usar o diagrama da Figura 28 para calcular a secção mínima necessária do cabo em função do comprimento para que as tensões mínimas sejam respeitadas.

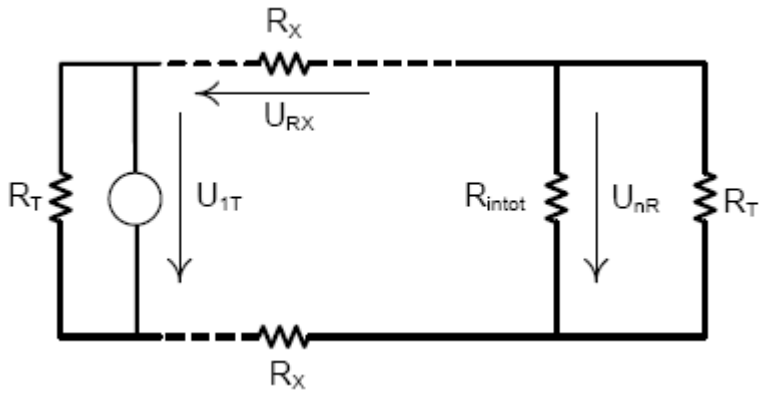


Figura 28: Resistências que normalmente se encontram num barramento CAN bifilar - adaptado de (Etshberger, 2001).

ρ - Resistividade linear do cabo (Ω/m);

L - Comprimento do barramento;

U_{1Tmin} - Tensão de *treshold* para o estado dominante do receptor (V);

U_{nRmin} – Tensão mínima de entrada necessária no nodo mais afastado;

U_{intot} – Resistência de entrada total dos *transceivers*.

R_T - Resistência de terminação (Ω);

Analisando o circuito da figura chegamos à seguinte fórmula:

$$A(L) \geq \frac{2 \cdot \rho \cdot L \cdot U_{nR\min}}{(U_{1T\min} - U_{nR\min}) \times R_{\text{int}ot} \parallel R_T}$$ (retirado de (Etshberger, 2001))

Para a resistência de terminação R_T, segundo a norma ISO 11898, deve ser usado um valor entre 108 e 132Ω, por isso, o cabo a ser utilizado para o barramento deve ter uma impedância característica à frequência de funcionamento dentro deste intervalo, caso contrário toda a linha fica desadaptada.

Dependendo do tipo de cabo utilizado para implementar o barramento, este vai ter várias características que vão influenciar o comportamento das formas de onda quando estas por ele "viajam". Como já vimos a impedância característica é uma delas, e esta depende da indutância e capacidade do cabo. Dependendo do valor desta indutância e capacidade, vamos ter diferentes velocidades dos sinais que "viajam" pelo cabo, na Figura 29 temos um modelo equivalente do barramento onde estão representadas as capacidades e indutâncias equivalentes.

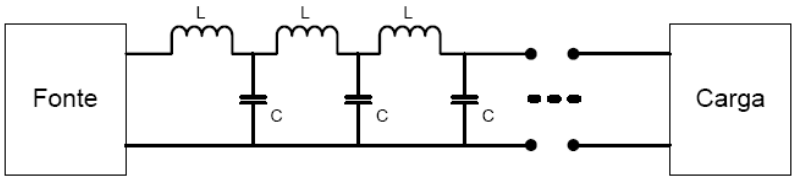


Figura 29: Circuito LC equivalente do barramento.

Então a velocidade do sinal no cabo será:

$$v = \frac{1}{\sqrt{LC}}$$

Com:

v -> [m/s]
L -> [H/m]
C -> [F/m]

Os valores de L e C variam com as dimensões transversais do cabo, por isso é necessário ter em atenção estes valores para a escolha do cabo utilizado para implementar o barramento. É também necessário ter em atenção que a linha deve estar o mais adaptada possível, ou seja, a impedância característica do cabo deve ser o mais próximo possível do valor da impedância de terminação. Caso contrário, a linha pode ficar desadaptada e as reflexões começam a deteriorar o sinal original. Podendo mesmo interferir com o sinal de tal forma que a margem

de segurança para que um nodo detecte um estado dominante do bit seja ultrapassada comprometendo assim toda a comunicação.

Num barramento CAN existem dois sinais, *CANHigh* e *CANLow*, Figura 30. Estes dois sinais fazem com que a transmissão seja feita de uma forma diferencial, o que confere ao barramento uma excelente imunidade ao ruído. Esta imunidade pode ser melhorada usando um cabo par entrançado com malha de protecção. As tensões características num barramento CAN dependem do estado em que este está no momento, recessivo ou dominante, como se pode ver na Figura 30 para o SN65HVD232D da *Texas Instruments*.

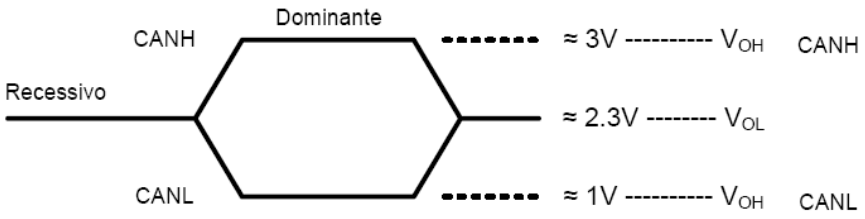


Figura 30: Voltagens características do estado recessivo e dominante do circuito SN65HVD232D – adaptado de (Instruments, 2006).

Capítulo 4

4. Tolerância a Falhas em Sistemas Distribuídos de Segurança Crítica

4.1. Introdução

Nos últimos anos têm vindo a ser propostos vários mecanismos, protocolos e extensões, tanto ao nível da camada física como de camadas superiores, para melhorar a tolerância a falhas nas redes de comunicação de sistemas de controlo distribuído. Por isso, sistemas ou protocolos que na sua forma nativa eram pouco tolerantes a falhas passam a ser mais seguros, sendo possível a sua aplicação em sistemas de segurança crítica. De seguida serão apresentadas algumas dessas abordagens, sendo na sua maioria baseadas no protocolo CAN.

4.2. *RedCAN*

O *RedCAN* é um mecanismo comercial que permite adicionar tolerância a falhas a um barramento CAN, quando estas ocorrem no nodo ou na ligação. É constituído por uma arquitectura de barramento em anel com capacidade para se auto reconfigurar. Este conceito utiliza interruptores configuráveis que permitem isolar do resto da rede um nodo ou um segmento de uma ligação impedindo o seu acesso ao barramento (ver Figura 32). Na Figura 31 está representada uma implementação do tipo Reino (Sivencrona, et al., 2004), baseada num protocolo de alto nível que utiliza o CAN. Esta implementação possui um mestre (*master*) ao qual se dá o nome de rei e possui também os escravos (*slaves*) designados por cidades que são controlados pelo mestre. Esta hierarquia possibilita uma implementação centralizada mas o mestre necessita de ter conhecimento sobre,

por exemplo, o número total de módulos *RedCAN* para ser capaz de decidir sobre a consistência do sistema.

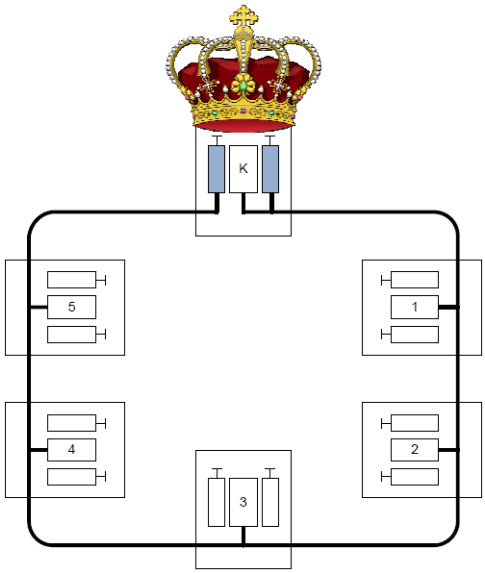


Figura 31: Arquitectura Reino para CAN com cinco cidades - adaptado de (Sivencrona, et al., 2004).

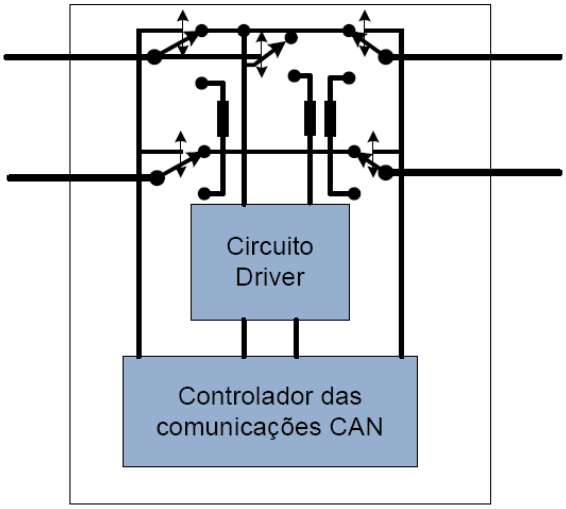


Figura 32: Módulo RedCAN no modo transparente - adaptado de (Sivencrona, et al., 2004).

Os autores de "*RedCAN: Simulations of two Fault Recovery Algorithms for* CAN" (Sivencrona, et al., 2004) propõem um algoritmo distribuído para alterar o protocolo centralizado original. Este algoritmo distribuído introduz redundância e aumenta a robustez do sistema. O algoritmo distribuído possui complexidade

logarítmica, ao contrário dos algoritmos centralizados que possuem complexidade linear com o aumento do número de nodos.

Quando é detectada uma falha numa ligação ou num nodo, o sistema isola-os da rede e define outro caminho para fazer a comunicação. Neste mecanismo o barramento é dividido em várias secções, sendo feita a ligação entre secções por intermédio de um módulo de comutação especializado. Estes módulos *RedCAN* são constituídos por relés e lógica de controlo que possibilitam fazer a terminação do cabo caso necessário, ou então permitem um funcionando num modo transparente. Este sistema ao utilizar circuitos com interruptores que podem estar abertos ou fechados faz com que se interrompam as comunicações não havendo assim continuação do serviço. A reconfiguração de uma rede em anel demora muito tempo, durante o qual a rede fica particionada. No caso do protocolo centralizado, o algoritmo de configuração utilizado pode demorar um segundo a configurar a rede. Com o novo algoritmo distribuído proposto pelos mesmos autores é possível obter melhores resultados que no protocolo centralizado. Estes resultados são apenas significativos quando o tamanho da rede aumenta. Na Figura 33 é possível apreciar os tempos de configuração necessários à medida que o número de nodos na rede vai aumentando, considerando que não existem erros na rede. Como se pode ver, o tempo necessário para configurar a rede no caso do protocolo centralizado cresce de forma linear, ao contrário do protocolo descentralizado em que o tempo cresce de forma logarítmica. Apesar da curva no caso do protocolo descentralizado ter valor médio superior ao centralizado, quando o número de nodos na rede cresce, o tempo necessário para configurar a rede aumenta muito pouco, enquanto no caso centralizado a curva continua a crescer de forma linear.

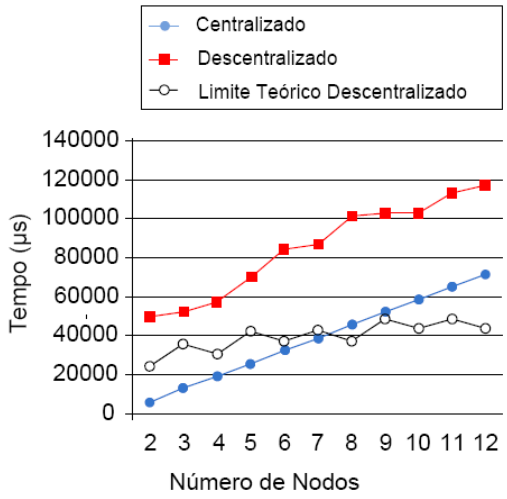


Figura 33: Tempos de arranque sem erros no sistema - adaptado de (Sivencrona, et al., 2004).

Na Figura 34 estão representados os tempos necessários para o sistema se reconfigurar para os dois tipos de protocolo. O protocolo descentralizado demora sempre o mesmo tempo independentemente do tipo de erro. No caso do protocolo centralizado o tempo cresce linearmente com o aumento de nodos na rede. Para uma rede com mais de doze nodos o tempo necessário para reconfigurar a rede para qualquer tipo de erro, no caso centralizado, é sempre superior ao caso distribuído.

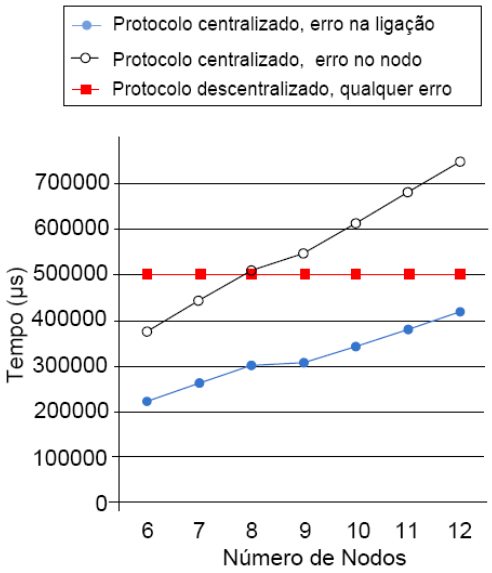


Figura 34: Tempos de reconfiguração na presença de erros (retirado de (Sivencrona, et al., 2004)).

Estes resultados foram obtidos através de simulação utilizando para isso o simulador *RedCAN simulation manager*.

Segundo os mesmos autores é possível fazer alterações a este sistema de forma a poder ser utilizado em aplicações que requerem redundância e um comportamento fiável. No entanto, é necessário fazer estudos para verificar se é possível utilizar este sistema em aplicações de segurança crítica. As razões para isto têm a ver com o facto de existirem limitações nos *switches* e circuitos que necessitam de testes para verificar a sua robustez. O algoritmo proposto necessita também de ser testado na prática para verificar se funciona como esperado.

4.3. *The Columbus' egg Idea for Bus Media*

O autor de (Rufino, 1997) propõem uma técnica muito simples para implementar redundância através de barramentos replicados. Este método tira vantagem de uma característica do CAN, a sua natureza *wired-AND*. É feita a replicação de barramentos (camada física) e dos meios de comunicação (interfaces), sendo única a camada MAC. Falhas que ocorram em meios de comunicação diferentes são consideradas independentes. Todos os meios de comunicação estão activos, logo a informação é igualmente transmitida em todos os barramentos replicados.

Caso ocorra o particionamento de um dos meios replicados, os nodos situados na partição que inclui o transmissor recebem um sinal correcto em todos os meios redundantes. Os nodos situados fora dessa partição recebem do meio em falha um sinal no estado recessivo, (estado lógico 1). Para resolver este problema é usada a função lógica *AND*, ligando os meios de comunicação replicados através desta função é então possível "produzir" um sinal correcto permitindo o normal funcionamento do nodo. Esta solução tem a desvantagem de apenas ser possível tolerar partições e falhas do tipo *stuck-at-recessive* nos

barramentos. Contudo existem soluções que permitem tornar esta solução menos restritiva no que toca à tolerância a falhas, (Rufino, 1997 p. página 5).

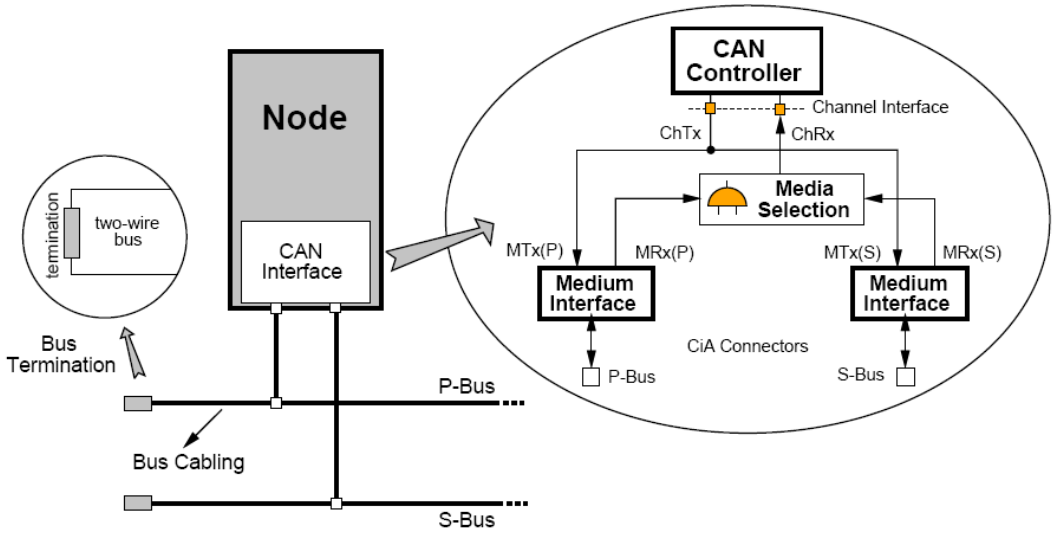


Figura 35: *The Columbus' egg idea for bus media* em CAN – retirado de (Rufino, 1997).

A replicação pode também ser feita ao nível da camada MAC, sendo assim possível adicionar redundância ao nível do nodo (ver Figura 36). Uma combinação da redundância ao nível do nodo e ao nível do barramento é assim possível, o que torna o sistema ainda mais tolerante a falhas.

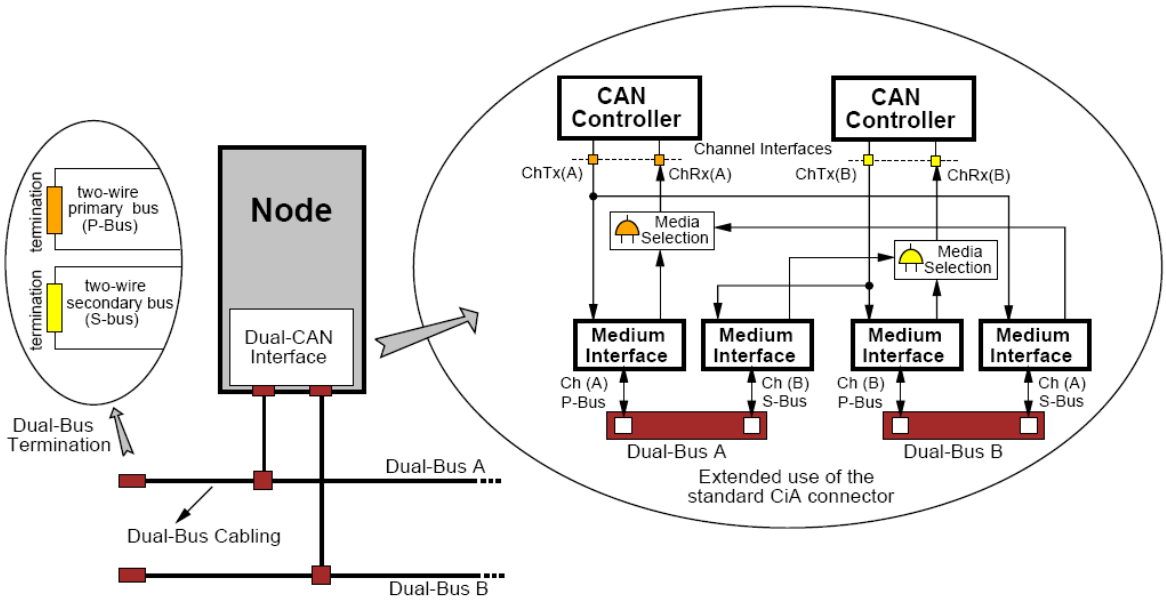


Figura 36: Dupla redundância, replicação da camada MAC e meio de transmissão - retirado de (Rufino, 1997).

Este método de aplicar redundância através de replicação é muito simples mas implica a alteração dos nodos, pois é necessário acrescentar-lhes uma camada de *software/firmware* para gerir a redundância nas comunicações e *hardware* para seleccionar o meio de transmissão a utilizar em caso de falha num desses meios. Numa rede CAN já previamente instalada seria necessário um esforço enorme para fazer uma actualização e implementar este tipo de replicação. Para além disso este método não permite aumentar a largura de banda do sistema, pois todos os barramentos replicados transmitem a mesma informação.

4.4. *CANcentrate* e *ReCANcentrate*

Para resolver o problema do fraco confinamento de erros que, por exemplo, a topologia do tipo barramento possui, foram propostas algumas topologias em estrela para o CAN. Os autores de "*CANcentrate: An active star topology for CAN networks*" (Barranco, et al., 2004) propuseram uma solução baseada num hub activo, com mecanismos melhorados de tratamento de falhas que permitem isolar da rede nodos em falha (ver Figura 37).

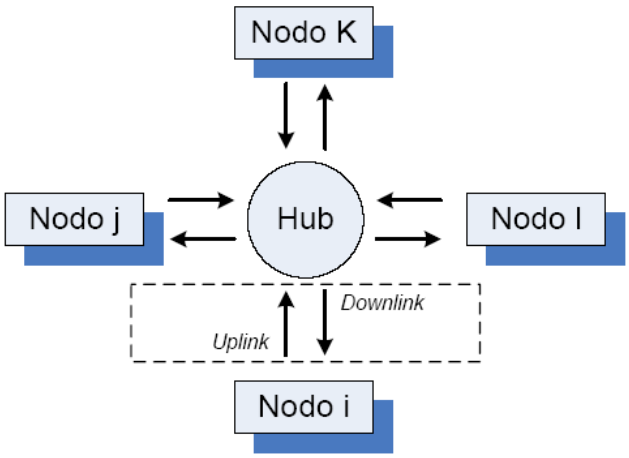


Figura 37: Arquitectura CANcentrate - adaptado de (Barranco, et al., 2004).

Este hub activo é compatível com o CAN original e permite detectar uma variedade de falhas, como por exemplo falhas *stuck-at* no nodo, corte do meio de comunicação, falhas *bit-flipping* e falhas nas ligações de acesso ao meio do nodo, que fazem com que o nodo em falha seja isolado quando é detectado o erro. A falha *stuck-at* ocorre quando um componente da rede (tanto um nodo como o meio) impõe um valor de bit constante, por exemplo, um nodo, que devido a um curto-circuito envia constantemente o valor lógico '0'. Como já foi referido antes, no CAN, o valor lógico '0' representa um bit dominante, enquanto que, o valor lógico '1' representa um bit recessivo. Uma vez que os bits dominantes sobrescrevem os bits recessivos, uma falha *stuck-at-dominant* força o meio a estar permanentemente no nível dominante. Uma falha *bit-flipping* ocorre quando um componente da rede envia permanentemente valores de bit errados, que corrompem qualquer sequência de bits a ser enviadas no barramento, por exemplo, uma má soldadura num componente.

Este hub foi o primeiro a ser projectado para o CAN para lhe fornecer tratamento de erros. No entanto este hub representa ele mesmo um ponto de falha. Uma solução proposta para este problema é a arquitectura documentada no artigo "*ReCANcentrate: A replicated star topology for CAN networks*" (Barranco, et al., 2005) que consiste na replicação do hub (ver Figura 38). O sistema passa então a ter uma topologia em estrela replicada usando para isso dois hubs CANcentrate.

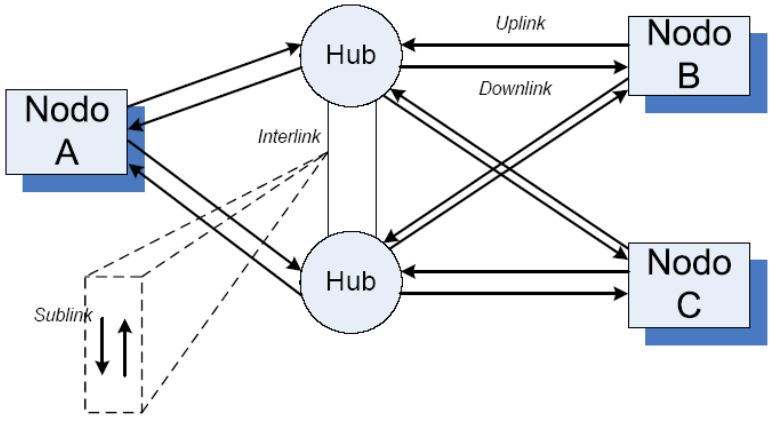


Figura 38: Arquitectura ReCANcentrate - adaptado de (Barranco, et al., 2005).

4.5. *FlexRay*

O *FlexRay* é um protocolo utilizado para a comunicação entre sistemas eléctricos nos automóveis e foi desenvolvido pelo consórcio *FlexRay Consortium*, que resulta da aliança entre vários fabricantes de semicondutores, sistemas electrónicos e automóveis. No ano 2000, a BMW e a *DaimlerChrysler* juntaram forças com a *Philips* e a *Motorola* para criar o *FlexRay Consortium*. Estes trabalham em conjunto para desenvolver um sistema de barramento com tolerância a falhas, determinismo e altas taxas de transmissão para aplicar em sistemas de controlo nos automóveis. Hoje em dia esta aliança conta já com vários fabricantes bastante conhecidos tais como, a *Fiat*, a *Ford*, a *Honda*, a *Renault*, a *Toyota*, a *Continental*, a *Fujitsu*, a *TDK*, a *Xilinx*, a *Mitsubishi Electric*, etc. (Consortium, 2000) Através da combinação da transmissão de mensagens escalonadas estaticamente e dinamicamente este protocolo oferece flexibilidade, possuindo assim as vantagens de protocolos síncronos e assíncronos. Dependendo das necessidades da aplicação, o ciclo de transmissão pode ser puramente síncrono, puramente assíncrono, ou uma mistura dos dois. A transmissão síncrona de dados possibilita comunicações *time triggered* e assim preencher os requisitos para a sua utilização em sistemas de segurança crítica. A transmissão assíncrona permite uma total utilização da largura de banda por parte dos nodos, para comunicações orientadas ao evento.

Este protocolo suporta também:

- Sincronização de relógio tolerante a falhas para uma base de tempo global, deste modo permite manter todas as tarefas agendadas pelos nodos, dentro de uma janela temporal predefinida;
- Acesso ao barramento livre de colisões. Um guardião do barramento (*bus guardian*) contribui para o confinamento de erros na camada física garantindo que não ocorrem colisões de dados mesmo que um controlador de comunicações falhe. O guardião do barramento monitoriza o tempo de forma independente. Se este encontrar um hiato no tempo, proíbe a transmissão dos dados;

- Latência das mensagens garantida. A transmissão síncrona de mensagens é determinística com um mínimo de latência nas mensagens e *jitter* garantido;
- Identificadores para orientar o endereçamento das mensagens. Para o caso de transmissões assíncronas, orientadas ao evento, o identificador é também utilizado para definir a prioridade da mensagem, como acontece com o CAN;
- Sistema de tolerância a falhas escalável via um meio de transmissão simples ou duplo, sendo assim possível utilizar dupla redundância.

Este protocolo suporta taxas de transferência de 10Mbit/s, com flexibilidade para facilitar a extensão do sistema e o uso dinâmico da largura de banda. A camada física suporta configuração flexível, ou seja, pode ser configurada para topologias em barramento ou em estrela. Nas topologias em estrela é possível utilizar estrelas activas (ver Figura 39). Neste exemplo estão ligados três tipos de rede entre si, uma composta por duas estrelas activas (sendo este o número máximo de estrelas activas que podem ser incluídas entre dois ECU), uma com estrela passiva e outra com topologia em barramento.

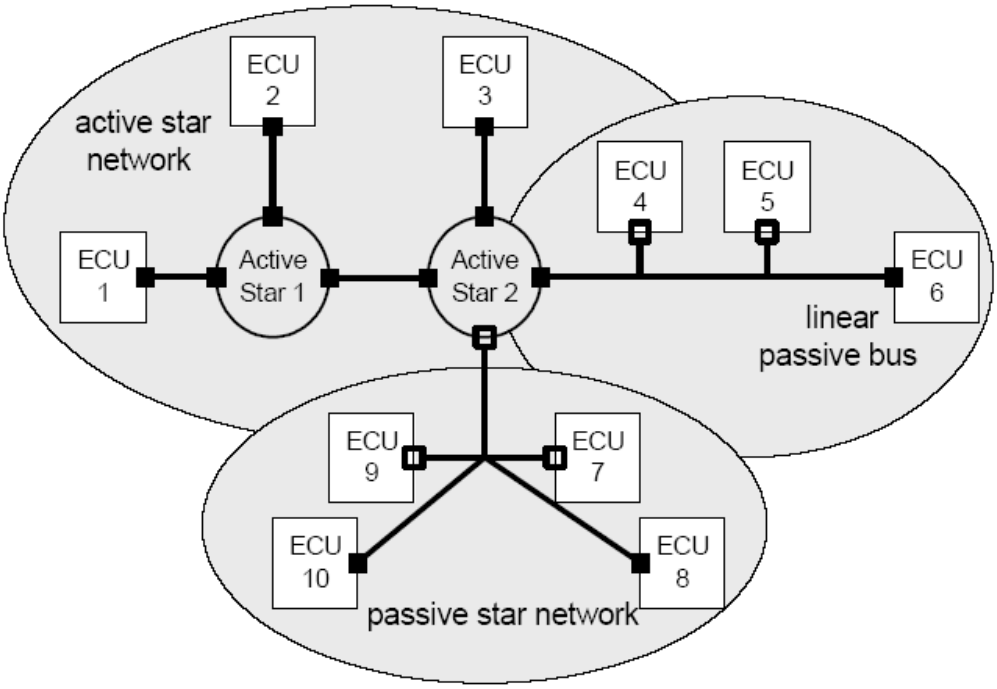


Figura 39: Exemplo de estrutura de barramento híbrida para *FlexRay* - retirado de (FlexRay, 2006).

Através dos dois canais disponíveis é possível obter redundância, pelo que o sistema pode continuar a funcionar mesmo quando um dos canais falha. As várias topologias apresentadas anteriormente funcionam com até um máximo de dois canais.

O controlo do acesso ao meio é baseado na repetição de um ciclo de comunicação. Num ciclo de comunicação existe a possibilidade de escolher entre dois esquemas de acesso ao meio. Estes esquemas são o (TDMA) que é estático e um esquema dinâmico baseado em *mini-slotting*. Na Tabela 3 é possível comparar algumas características entre o *FlexRay* e o CAN na sua forma mais simples.

	CAN	*FlexRay*
Canais	Único	Único/Duplo
Velocidade	≤1 Mbit/s	10 Mbit/s
Time Triggered	Não	Sim
Arbitragem	CSMA	TDMA
Topologias Suportadas	Barramento	Barramento e Estrela
Aplicável a sistemas de controlo de segurança crítica?	Não	Sim
Custo da implementação	Baixo	Elevado (comparado com o CAN)

Tabela 3: Comparação entre *FlexRay* e CAN, ambos na sua forma nativa – adaptado de (Consortium, 2000).

Uma redução no custo de produção do *FlexRay* aliado às suas características funcionais que permitem a sua utilização em aplicações de segurança crítica poderá fazer com que o *FlexRay* seja aplicado em gerações futuras de sistemas de segurança crítica nos automóveis.

4.6. *FlexCAN*

O *FlexCAN* foi projectado para suportar aplicações de segurança crítica como as que são utilizadas nos veículos. É constituído por uma nova arquitectura *FlexCAN* e um protocolo associado *SafeCAN* (R. Pimentel, et al., 2004). O protocolo *SafeCAN* é adicionado ao protocolo CAN nativo para que este possa ser aplicado em sistemas onde são necessários rigorosos níveis de confiança. Para além de alguns atributos importantes que dão ao CAN um certo grau de confiança, o *FlexCAN* adiciona mais alguns atributos, (R. Pimentel, et al., 2004):

- Gestão de redundância. Detecção de erros nos nodos e gestão de um conjunto de nodos replicados;
- Gestão de canais CAN replicados;
- Gestão de uma FTU;
- Número de sequência. Cada mensagem possui um número de sequência no primeiro byte de dados. Este byte possui informação sobre o tipo de protocolo, sobre a FTU, sobre o número de série (este número é utilizado para seleccionar a réplica a ser utilizada na FTU) e sobre o IG (mensagem *inter-group* ou *intra-group*);
- Mensagens temporizadas;
- Sincronização orientada à aplicação, (sensores, actuadores, controladores, processo sincronizado no tempo);
- Gestão da aplicação. Regista o funcionamento de várias aplicações;
- Sincronização temporizada, iniciada pelo produtor da mensagem;
- Gestão de omissões, falhas e erros;
- Comportamentos que suportam falha-silêncio nos nodos;
- Comportamentos que suportam falhas nos canais CAN;
- Comportamentos que permitem à aplicação falhar em segurança.

O protocolo *SafeCAN* permite ao sistema gerir todos os dispositivos replicados, este protocolo foi colocado fora das camadas do modelo OSI, numa camada de segurança (*safety layer*) (ver Figura 40).

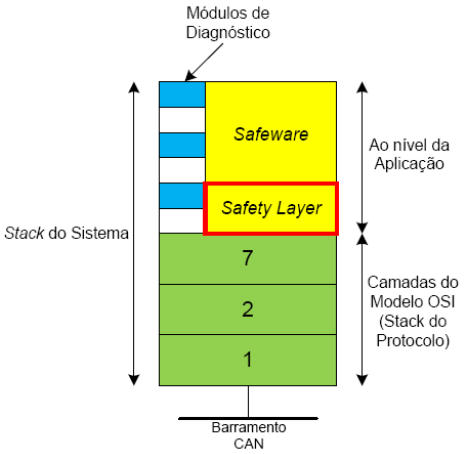


Figura 40: *Safety Layer* e *Safeware* sobre a *stack* do modelo OSI - adaptado de (R. Pimentel, et al., 2004).

A replicação permite adicionar segurança, disponibilidade e fiabilidade. No *FlexCAN*, são replicados dois componentes, os canais de comunicação e os nodos (ver Figura 41). A FTU - *Fault Tolerant Unit* da figura possui um nodo e duas réplicas. Como se pode ver na figura, cada nodo está ligado a três canais CAN, o que resulta numa rede com nove interfaces. Para simplificar a gestão dos componentes replicados é utilizado o protocolo *SafeCAN*. O *SafeCAN* faz com que a rede veja os nodos replicados como um só.

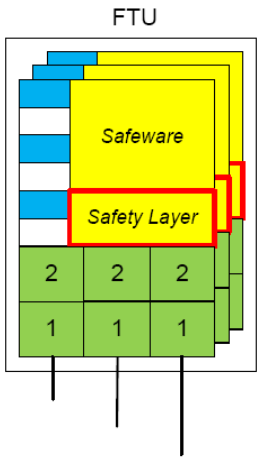


Figura 41: Componente FTU (*fault tolerant unit*) utilizado pelo protocolo SafeCAN - adaptado de (R. Pimentel, et al., 2004).

Na Figura 42, está representado um exemplo de uma aplicação. Esta contem três FTUs, um gestor da rede (*Network Manager*) e duas aplicações convencionais que podem coexistir numa rede de segurança crítica.

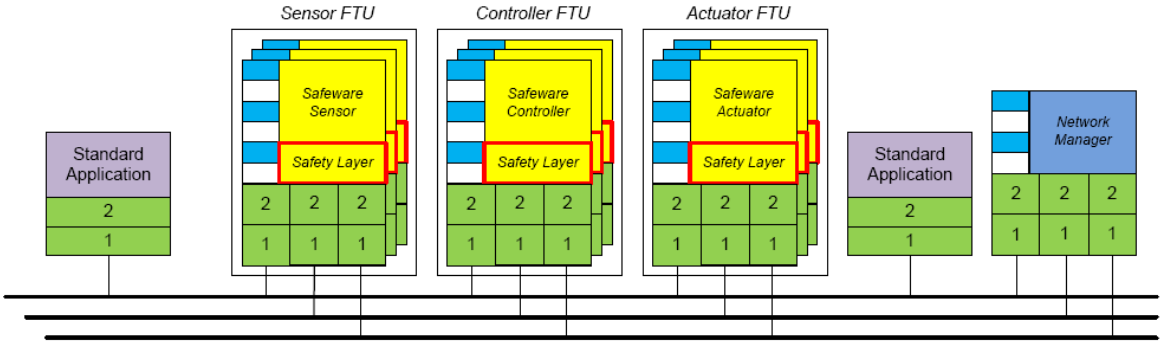


Figura 42: Exemplo de aplicação envolvendo componentes replicados - adaptado de (R. Pimentel, et al., 2004).

4.7. *Dynamic Topology Management* em CAN

Os autores do artigo "*Dynamic Topology Management in CAN*" (Silva, et al., 2006) propõem dois tipos de componentes, uma arquitectura e protocolos para permitir a gestão dinâmica da topologia de redes com redundância baseadas em CAN. Este sistema permite a utilização de M barramentos replicados que permitem simultaneamente um aumento na largura de banda e uma reconfiguração da rede em caso de falha num dos barramentos. Com este sistema é possível gerir o uso de tolerância a falhas e largura de banda para controlar a transmissão redundante de mensagens, para reagir a falhas no barramento mantendo uma redundância com tantos níveis quantos os barramentos em ausência de falhas disponíveis. Em caso de falha num barramento é possível utilizar os barramentos disponíveis e ausentes de falhas para transmitir tráfego não crítico, ficando assim o sistema a funcionar num modo de operação degradado. Todo este sistema é possível de ser implementado na infra-estrutura de rede, não sendo portanto necessário modificar os nodos já existentes no sistema distribuído.

A arquitectura proposta pode ser vista na Figura 43. Esta arquitectura faz uso de dois tipos de componentes, uma NSU e uma TMU que serão relatados com maior pormenor nas subsecções 5.1 e 5.2 respectivamente.

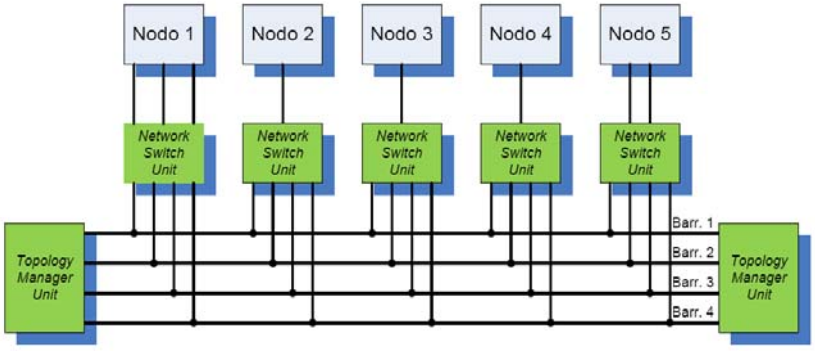


Figura 43: Arquitectura de rede proposta DTM - adaptado de (Silva, et al., 2006).

4.8. Comparação das Várias Abordagens Apresentadas

Todos os sistemas apresentados neste capítulo, excepto o *FlexRay* uma vez que não é baseado em CAN, permitem melhorar a capacidade de tolerância a falhas do CAN nativo. Devido à grande importância do *FlexRay*, este não podia deixar de ser referenciado. Na Tabela 3 é possível ver a comparação de algumas das suas características com o CAN nativo, sendo o *FlexRay* melhor no que toca a tolerância a falhas. No entanto os sistemas aqui apresentados baseados em CAN fazem com que esta diferença não seja assim tão significativa. Todos eles acrescentam redundância, mas nem todos de uma forma eficaz. Apenas o sistema DTM - *Dynamic Topology Management* em CAN (Silva, et al., 2006), permite adicionar redundância de uma forma eficiente pois permite uma melhor gestão da largura de banda.

	RedCAN	***The Columbus' egg Idea for Bus Media***	***(Re)CANcentrate***	***FlexCAN***	***DTM***	***FlexRay***
Replicação (Canais, Nodos)	Canal-Não Nodos-Não	Canal-Sim Nodos-Sim	Canal-Não Nodos-Não	Canal-Sim Nodos-Sim	Canal-Sim Nodos-Sim	Canal-Sim Nodos-Não
Velocidade	≤1 Mbit/s	≤1 Mbit/s	≤1 Mbit/s	≤1 Mbit/s	≤1 Mbit/s	10 Mbit/s
Time Triggered	Não	Não	Não	Não	Não	Sim
Arbitragem	CSMA	CSMA	CSMA	CSMA	CSMA	TDMA
Topologias Suportadas	Anel (mais usual, embora outras topologias possam ser usadas)	Barramento	Estrela	Barramento	Barramento e Estrela Virtual	Barramento e Estrela
É Necessário Alterar os Nodos?	Sim	Sim	Não	Sim	Não	Não

Tabela 4: Comparação de algumas características das abordagens apresentadas.

Capítulo 5

5. Gestão Dinâmica da Topologia em CAN

Neste capítulo será feita uma apresentação dos componentes que constituem o sistema e que permitem fazer uma gestão dinâmica da topologia em CAN. O primeiro componente apresentado é a NSU - *Network Switch Unit*, este componente é responsável pelas ligações físicas dos nodos aos barramentos replicados. De seguida será feita a apresentação da TMU – *Topology Manager Unit*, este componente é responsável pela gestão dinâmica da topologia. Esta gestão é feita através do envio de comandos às NSUs para assim configurar as ligações dos nodos aos barramentos replicados de acordo com as necessidades. Estes comandos serão apresentados neste capítulo onde é feita uma descrição do seu funcionamento, da codificação e do seu mapeamento em tramas CAN. É também feita uma apresentação das falhas suportadas pelo sistema, ou seja, é possível que o sistema continue a funcionar na presença de falhas mas num modo degradado de serviço. Este sistema pode operar de várias formas, neste capítulo é feita uma apresentação dos vários cenários de operação possíveis.

5.1. *Network Switch Unit* (NSU)

A NSU - *Network Switch Unit* é responsável pela ligação física entre os nodos do sistema e os barramentos CAN disponíveis. Uma vez que a NSU liga as N interfaces do nodo aos M barramentos disponíveis, assume portanto a forma duma matriz NxM. A arquitectura interna da NSU está representada de uma forma muito simplificada na Figura 44, como se pode ver nesta figura a NSU é composta por uma *switch matrix*, um *switch controller* e uma *switch table*. A *switch matrix* faz a ligação física entre os nodos e os barramentos e é também responsável pela comutação entre barramentos replicados. Como irá ser visto nos capítulos seguintes a *switch matrix* será composta por interruptores analógicos que serão

responsáveis pelas ligações físicas entre os nodos e os barramentos. A *switch table* armazena a configuração das ligações a serem feitas pela *switch matrix*. Esta tabela é composta por M colunas e N linhas, em que cada coluna representa as ligações aos barramentos CAN e cada linha representa as ligações ao nodo (*transceivers* CAN). O *switch controller* é responsável pelo controlo de todas as operações realizadas pela NSU, por exemplo, controla o abrir e fechar dos interruptores (*switches*) da *switch matrix*, escreve na *switch table* o estado dos *switches*, envia mensagens de resposta para a TMU etc.

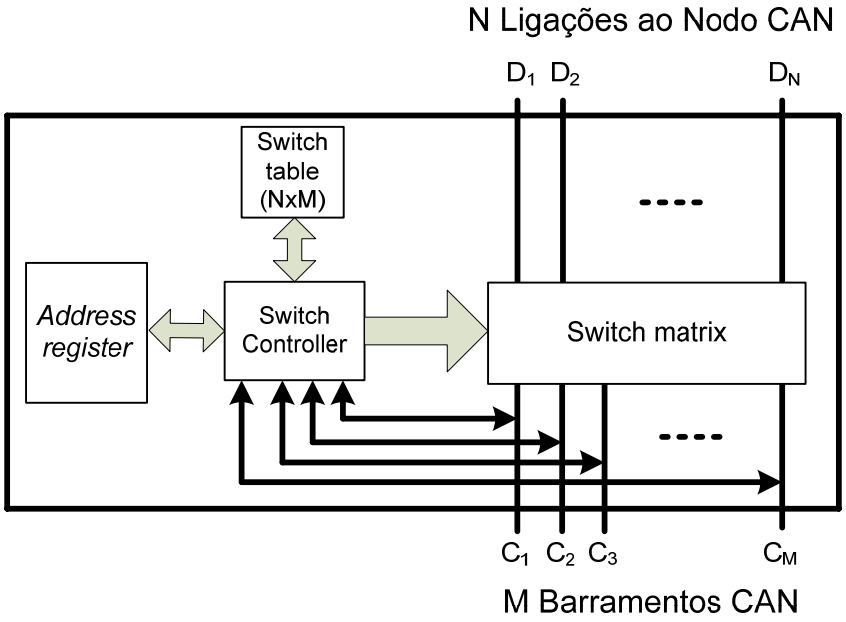


Figura 44: Arquitectura interna da NSU - *Network Switch Unit* - adaptado de (Silva, et al., 2006).

No exemplo representado na Figura 45, o projectista do sistema pode escolher usar uma ligação onde o nodo 1 é ligado aos barramentos 1, 2 e 3, o nodo 5 é ligado aos barramentos 1 e 2, e os nodos 2 e 3 são ligados ao barramento 1 e o nodo 4 é ligado ao barramento 3.

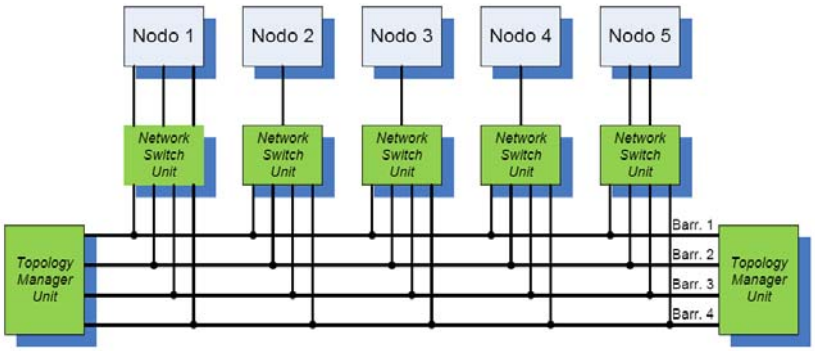


Figura 45: Exemplo de uma configuração utilizando a DTM - adaptado de (Silva, et al., 2006).

Com esta configuração obtemos a seguinte tabela da NSU:

Nodo	Interruptores	Barramentos em uso
Nodo 1	1,1,1,0	1,2,3
Nodo 2	1,0,0,0	1
Nodo 3	1,0,0,0	1
Nodo 4	0,0,1,0	3
Nodo 5	1,1,0,0	1,2

Tabela 5: Tabela da NSU - adaptado de (Silva, et al., 2006).

Com esta configuração o barramento 4 é apenas usado com a finalidade de obter redundância, isto significa que, se um dos outros barramentos falhar o sistema pode ser reconfigurado para passar a usar o barramento 4. A *switch table* armazenada na NSU pode ser lida e escrita pelo *switch controller*. O *switch controller* recebe mensagens especiais enviadas pela TMU através dos barramentos CAN e que alteram a tabela e a matriz de acordo com as necessidades. O *switch controller* deve ter um controlador CAN por cada barramento e é responsável por responder a mensagens especiais enviadas pela TMU para verificar o estado da NSU e dos barramentos. O registo do endereço (*address register*) armazena o endereço identificador da NSU.

5.2. *Topology Manager Unit* (TMU)

Na Figura 46 está ilustrada a arquitectura da TMU - *Topology Manager Unit*.

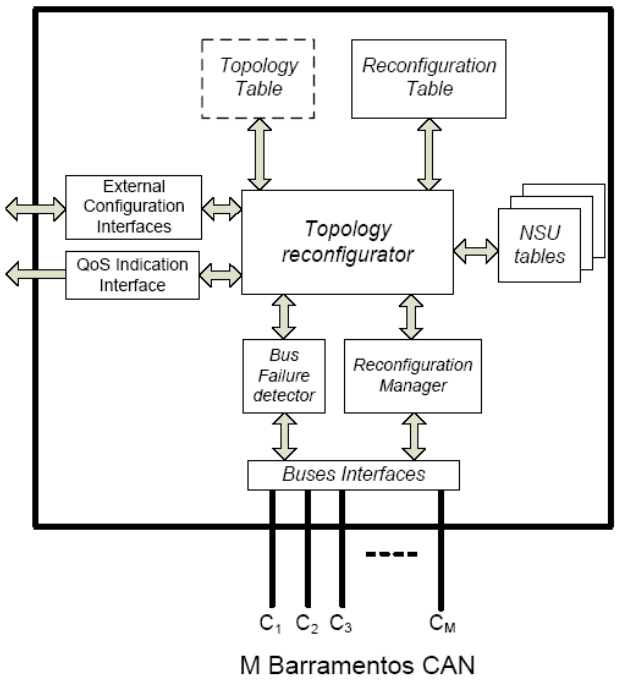


Figura 46: Arquitectura da TMU - *Topology Manager Unit* - adaptado de (Silva, et al., 2006).

A TMU é composta por vários componentes que operam usando a informação armazenada em tabelas. O componente mais importante, que toma as decisões necessárias no que toca à gestão dinâmica da topologia é o TR - *Topology Reconfigurator*. Quando é detectada uma falha no barramento o TR começa a funcionar utilizando a informação vinda do BFD - *Bus Failure Detector*. Essa informação pode resultar de eventos espontâneos, por exemplo, a detecção de falhas *stuck-at* num dos barramentos, ou depois de uma verificação periódica do estado do barramento onde é detectada uma partição. Para prevenir a ocorrência de omissões de mensagens, o barramento é considerado permanentemente em falha quando o respectivo controlador CAN da TMU atinge o estado de erro passivo. O protocolo de detecção de partições será detalhado na subsecção seguinte.

O TR reage a falhas no barramento de acordo com os procedimentos armazenados na tabela de reconfiguração, RT - *Reconfiguration Table*. Estes procedimentos definem o que deve ser feito no caso de haver uma falha num barramento.

Para o caso de 4 barramentos redundantes existem 14 combinações possíveis de falha no barramento para os quais o TR tem que lidar, (Silva, et al., 2006). No entanto, é necessário ter em conta que algumas destas combinações levam a um estado onde a operação do sistema é impossível. Nestes casos a RT deve ter informação para parar o sistema. A RT contém também uma configuração genérica (armazenada na RT aquando do seu fabrico), que é utilizada após uma falha temporária podendo assim o sistema retomar o seu correcto funcionamento. A TMU faz a gestão das ligações físicas de uma forma autónoma, não sendo preciso portanto outros elementos no sistema. A TMU possui uma interface com o exterior, a *QoS Indication Interface*, que indica o estado da operação através da utilização da informação disponível na tabela de reconfiguração em uso.

Como já foi referido, a tabela de reconfiguração possui todas as configurações das NSUs. Sempre que uma destas configurações é alterada o TR activa o gestor de reconfiguração que envia a nova configuração para as NSU. As tabelas NSU (NSU *tables*) contêm o estado actual de todos os interruptores presentes na rede. A identificação das NSUs é também ai armazenada. Para possibilitar a configuração da TMU, esta possui uma interface usada para programar a RT.

A TMU pode ser implementada em hardware, sendo também possível utilizar um processador com uma capacidade de processamento adequada.

5.2.1. Protocolo de Replicação da TMU

Como a TMU representa um ponto de falha na rede, necessita de ser replicada para assim as comunicações poderem continuar em caso de falha de uma delas. Com este mecanismo as TMUs seguem um comportamento líder/seguidor (*leader/follower*), onde o líder questiona periodicamente qual é o estado dos barramentos utilizando comandos específicos. O seguidor tenta fazer o mesmo ao mesmo tempo, mas se o líder transmitir o comando com sucesso o seguidor aborta a transmissão e recebe o comando do líder. Se o líder falhar na transmissão da mensagem de sincronização, o seguidor torna-se o líder. Este mecanismo tem a vantagem de ser simples e de permitir detectar partições nos barramentos. Deste modo, o líder e o seguidor trocam mensagens de como estão a ver o estado dos barramentos e comparam-nas para ver se há diferenças indicando se há partições ou outro tipo de falhas. Apenas o líder detecta as falhas, o seguidor envia a sua visão do estado dos barramentos mas nunca decide se um barramento em particular está em falha. A seguir à detecção, o líder altera a configuração da topologia, informando o seguidor da nova configuração. Todos os comandos são enviados em paralelo por todos os barramentos, usando mensagens de alta prioridade.

5.3. Tipos de Falhas Suportados pelo Sistema

Os tipos de falhas às quais a arquitectura DTM – *Dynamic Topology Management* proposta por (Silva, et al., 2006) será tolerante são as seguintes:

- Falhas no nodo;
- Falhas do tipo transiente no meio de transmissão;
- Atomicidade das mensagens;
- Falhas permanentes no meio de transmissão;
- Partições no barramento.

5.3.1. Falhas no Nodo

Tanto o nodo como a *switch unit* são uma unidade tolerante a falhas (FTU). Esta FTU é tolerante a falhas do tipo falha-silêncio tanto no domínio do tempo como no domínio do valor. A TMU – *Topology Manager Unit* é também tolerante a falhas do tipo falha-silêncio tanto no domínio do tempo como no domínio do valor. Este tipo de comportamento nestes componentes pode ser conseguido através de replicação, onde tanto o valor como o tempo das saídas das réplicas devem coincidir. Para adicionar este tipo de comportamento a estes componentes, podem ser utilizadas algumas técnicas adoptadas em (Joaquim Ferreira, 2006) para o caso do FTT-CAN.

5.3.2. Falhas do Tipo Transiente no Meio de Transmissão

Este tipo de falhas pode ocorrer no meio de comunicação devido a interferências vindas do meio exterior, como por exemplo as interferências electromagnéticas. Só serão consideradas falhas que alterem o valor de pelo menos um bit.

5.3.3. Atomicidade das Mensagens

Quando uma mensagem é transmitida em paralelo por vários barramentos replicados e é recebida por um nodo com pelo menos duas interfaces, a sua transmissão é considerada atómica. Ou seja, considera-se que a mensagem recebida pelo nodo é igual nas duas interfaces. Este pressuposto é baseado em dados experimentais que provam que o rácio de erros nos bits CAN num ambiente agressivo pode ser de 2.6×10^{-7} com a correspondente rácio de omissão inconsistente de mensagens de 10^{-9} por hora (Silva, et al., 2006).

5.3.4. Falhas Permanentes no Meio de Transmissão

Uma vez que no CAN o meio de transmissão é um ponto de falha, falhas permanentes podem ocorrer, tais como partições no barramento ou falhas *stuk-at.*[3] O tipo de falha *stuk-at* pode ocorrer quando um componente que por algum motivo começa a transmitir para o barramento um valor de bit fixo, o que compromete todas as comunicações futuras. Na Figura 47, caso A, está representado um exemplo deste tipo de falha.

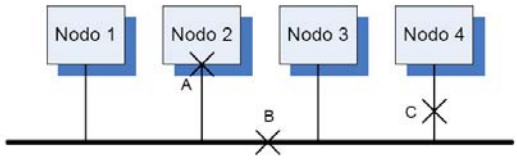


Figura 47: Exemplos de algumas falhas possíveis na topologia barramento - (adaptado de (Manuel Barranco, 2004))

Para prevenir a ocorrência de omissões de mensagens, o barramento é considerado como estando permanente em falta sempre que o respectivo controlador CAN localizado na TMU atinge um estado de erro passivo.

5.3.5. Partições no Barramento

Este tipo de falha está representada na Figura 47, caso B, onde um possível corte no meio de comunicação divide o barramento, criando assim uma partição, isolando os nodos um e dois dos nodos três e quatro.

O facto da arquitectura proposta em (Silva, et al., 2006) possuir duas TMU, (uma é a réplica da outra e estão as duas situadas nas extremidades do barramento), facilita a detecção de partições no barramento. No entanto, se a réplica que está activa falhar, o sistema perde a capacidade de detectar este tipo de falha e o sistema passa a funcionar num modo degradado.

[3] Falhas *stuk-at* ocorrem quando é imposto um valor fixo ao barramento, que pode ser tanto o valor lógico '0' como '1'.

5.4. Cenários de Operação

Na Figura 48 está representada, num exemplo simples, uma possível implementação da gestão dinâmica de topologia num sistema em que neste caso cada nodo possui apenas um controlador CAN, não existindo por isso redundância ao nível do nodo. O sistema possui quatro barramentos, existindo assim redundância quádrupla ao nível do barramento. Se um barramento por algum motivo falhar, a TMU envia comandos para as NSU trocarem de barramento para poderem continuar a fazer a comunicação. Durante esta operação de troca de barramento os nodos vão estar desconectados da infra-estrutura de comunicação durante um intervalo de inacessibilidade que equivale ao tempo necessário para detectar a falha mais o tempo necessário para fazer a troca de barramento.

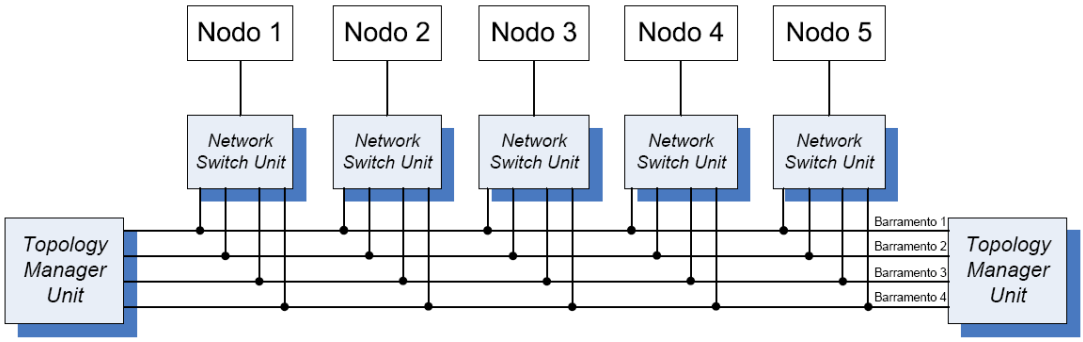


Figura 48: Exemplo com topologia dinâmica - adaptado de (Silva, et al., 2006).

Se este tempo de inacessibilidade não for tolerável para a aplicação, então o *switch* 1:4 representado na Figura 48 não chega, sendo necessário o uso de nodos com replicação. A solução proposta oferece uma matriz N:M, ou seja, um *switch* N por M que permite a conexão de nodos mais complexos na camada física. Na Figura 49, como exemplo, está representada uma configuração com *switchs* 2:4, onde os nodos 1 e 5 possuem ligações replicadas.

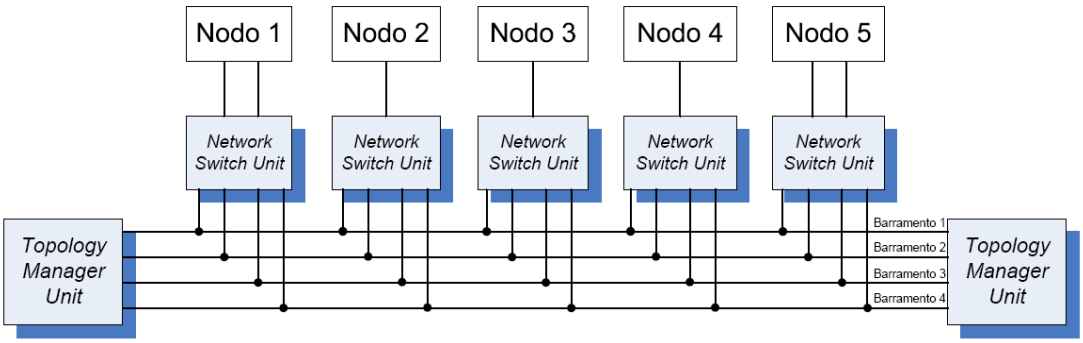


Figura 49: Exemplo com NSUs 2:4 - adaptado de (Silva, et al., 2006).

De notar que esta solução resolve apenas o problema da replicação da camada física, pois redundância das NSUs não é considerada. Uma falha numa NSU equivale a uma falha no nodo pois este perde acesso aos barramentos.

A arquitectura proposta pode ser usada para definir um barramento dedicado para a comunicação entre dois nodos. Esta característica é importante para providenciar largura de banda permanente ou temporária para aplicações mais exigentes sem interferir com o restante tráfico. Outra característica desta arquitectura é a possibilidade de utilizar caminhos virtuais, como se pode ver na Figura 50, o nodo 1 comunica com o nodo 4, o nodo 2 comunica com o nodo 3 e o nodo 5 comunica com o nodo 6. Podendo esta configuração ser temporária ou não. É possível implementar uma estrela virtual, sendo necessário um nodo central com uma interface por cada nodo com que vai comunicar, é possível ver um exemplo na Figura 51.

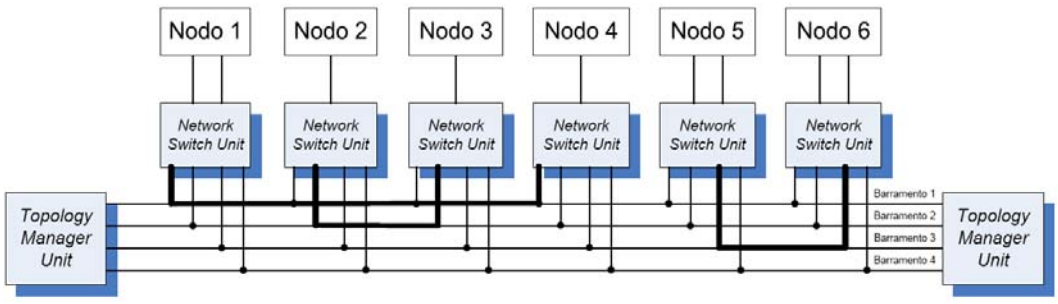


Figura 50: Exemplo com caminhos virtuais - adaptado de (Silva, et al., 2006).

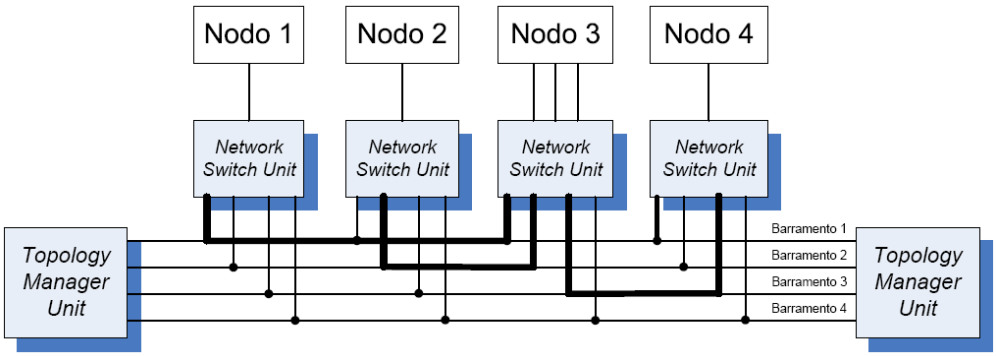


Figura 51: Exemplo com estrela virtual - adaptado de (Silva, et al., 2006).

Neste caso como temos quatro barramentos disponíveis, o barramento quatro pode ser usado para redundância, no caso de falha num dos outros barramentos, sendo o tráfico redireccionado para este.

5.5. Comandos do Sistema

O sistema proposto possui vários comandos que gerem e configuram a topologia. Estes comandos são enviados pela TMU a todas as NSUs através dos barramentos disponíveis. As NSUs podem ou não enviar uma resposta dependendo dos comandos. As mensagens enviadas pelas TMUs são chamadas de comandos e as mensagens enviadas pelas NSUs são chamadas de respostas. Uma NSU só envia uma resposta se tiver recebido um comando vindo da TMU.

Serão usados dois tipos de comandos para configurar o sistema e para enviar informação durante a operação: comandos de Descoberta (*Discovery Commands*) e comandos de Configuração (*Configuration Commands*). Comandos de Operação (*Operational Commands*) serão usados para programar e verificar o estado das *switch matrix* das NSUs.

5.5.1. Comandos de Descoberta e de Configuração

Como as NSUs vão funcionar em rede torna-se necessário atribuir-lhes um endereço. As NSUs são configuradas com um endereço aquando da sua fabricação, sendo este endereço igual para todas as NSUs. Este endereço pode ser alterado através de um comando de configuração enviado pela TMU. Assim sendo, as NSUs terão que ser ligadas ao sistema uma de cada vez e só depois de alterado o seu endereço de fábrica é que se pode ligar uma nova NSU. O comando utilizado para configurar o endereço é o seguinte:

SET_ADDRESS (TNadd, NNadd, ACK, bus1, …, busM)

TNadd - endereço da NSU alvo;
NNadd - novo endereço atribuído à NSU;
ACK - é uma *flag* que indica se a TMU requer uma resposta ou não. Esta resposta pode ser enviada por um barramento específico requerido pela TMU e para indicar qual o barramento a ser utilizado basta colocar a '1' uma das *flags bus#*. É possível então que a TMU controle a transmissão da resposta por mais do que um barramento disponível. Isto pode ser usado no futuro para fazer o diagnóstico do sistema.

A resposta da NSU, quando requerida, deverá ter a forma seguinte:

ACKNOWLEDGE (Nadd, CCode)

Nadd - endereço actual da NSU;
CCode - indica qual é o comando que está a ser confirmado.

Existe ainda um comando enviado pela NSU para a TMU que tem como função indicar à TMU que foi incluída uma nova NSU na rede. Assim sendo, sempre que

uma nova NSU é inserida na rede é enviado este comando para que a TMU tome conhecimento da sua presença e inicie assim o procedimento de configuração. Este comando, ***NEW_NSU***, não possui informação no campo de dados e tem como código uma sequência de três zeros "000". A única informação relevante na mensagem CAN deste comando está presente no campo ID, onde está presente o endereço de "fábrica" da NSU. A TMU reconhece esse endereço e inicia a configuração da NSU.

5.5.2. Comandos de Operação

Os comandos de operação serão usados pela TMU para reconfigurar as NSUs em caso de falha, ou então, para configurar uma nova NSU inserida na rede. As operações utilizadas para ligar ou desligar os nodos do barramento são feitas de forma assíncrona, para tal são usados pela TMU dois comandos para ligar ou desligar nodos dos barramentos e para sincronizar as ligações ou, pelo menos, para controlar o instante de ligação.

O comando seguinte é usado pela TMU para isolar um nodo ou um conjunto de nodos dos barramentos através da abertura de todos os *switches* da *switch matrix* dos nodos alvo. O comando tem a seguinte forma:

OPEN_SWITCHES (Nadd, tbus1, …, tbusM, ACK, rbus1, …, rbusM)

Nadd - é o endereço da NSU;
tbus# - são *flags* que indicam os barramentos alvo dos quais os *switches* devem ser desligados;
ACK - é uma *flag* que indica se é necessário enviar resposta. A resposta, ou confirmação, é enviada pelos barramentos indicados pelas *flags* rbus#.

O comando que restabelece as ligações definidas na *switch table* da NSU tem a seguinte forma:

CLOSE_SWITCHES (Nadd, tbus1, ..., tbusM, ACK, rbus1, ..., rbusM)

Este comando permite estabelecer a sincronização de uma nova configuração se for requerido com um endereço de *broadcast*.

Se for necessário reconfigurar o sistema, a TMU envia um comando que escreve uma nova *switch table* na NSU, o comando é o seguinte:

MATRIX_WRITE (Nadd, ACK, bus1, ..., busM)

O comando que permite ler a *switch table* da NSU é o seguinte:

MATRIX_READ (Nadd, rbus1, ..., rbusM)

A NSU responde com:

MATRIX_DATA (Nadd, switches_data)

5.5.3. Mapeamento dos Comandos em Tramas CAN

O mapeamento destes comandos em tramas CAN é feito usando o endereço *Nadd* mais um bit no campo *ID* da trama que diferencia a fonte da mensagem, TMU ou NSU. Ou seja, se o *Nadd* da NSU for por exemplo "000001", então é adicionado mais um bit no estado lógico '1' a *Nadd* ficando então "0000011". Para o caso da TMU este bit é adicionado no estado lógico '0'. Isto facilita o envio de mensagens de *unicast*, *multicast* ou *broadcast*, programando a máscara adequada da NSU. O código e os restantes dados do comando ou resposta estão inseridos no campo de dados da trama CAN (ver Figura 52).

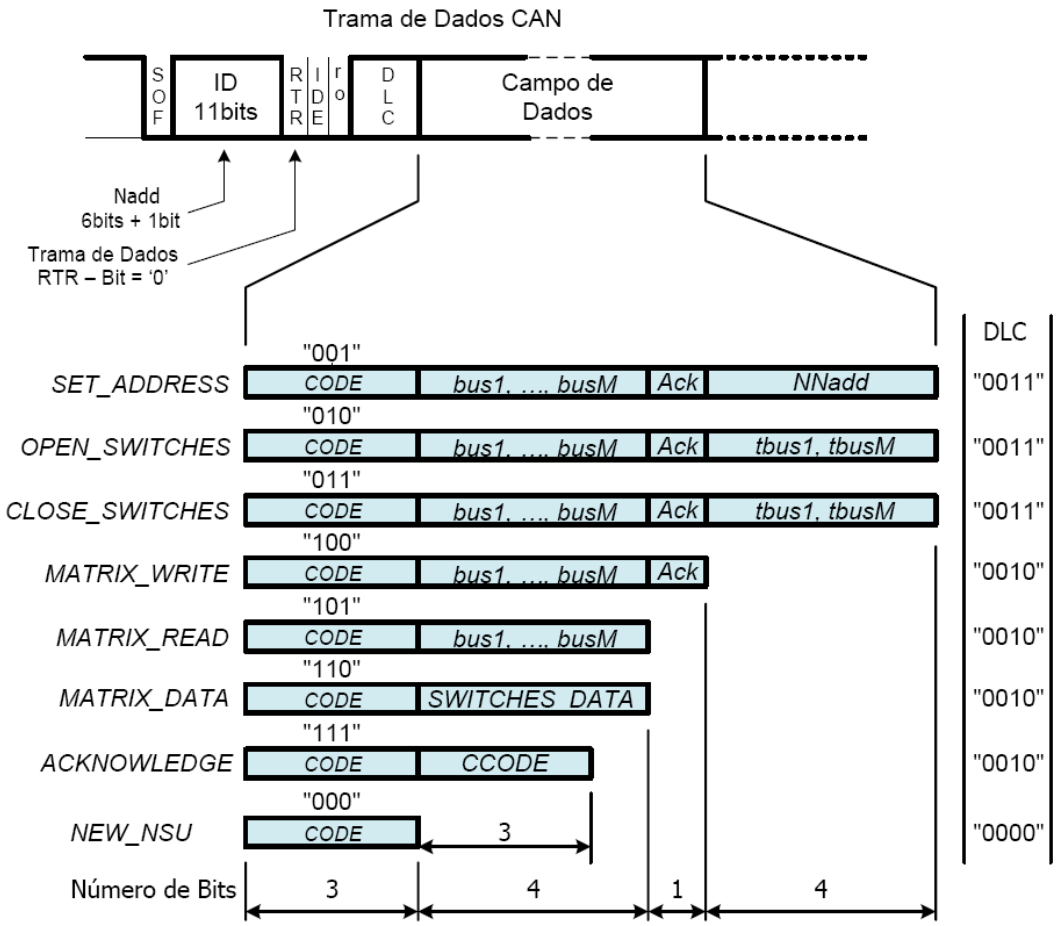


Figura 52: Mapeamento dos comandos no campo de dados das mensagens CAN.

O campo *CODE* alberga o código do comando, para codificar os sete comandos foram usados três bits. O campo que se segue a este possui quatro bits, excepto no comando *ACKNOWLEDGE* que possui neste caso três bits, o mesmo número de bits do campo *CODE*. Cada um destes quatro bits representa um barramento dependendo o seu tamanho do número de barramentos redundantes, excepto nos comandos *MATRIX_DATA* e *ACKNOWLEDGE* em que este campo possui uma função diferente. Os bits deste campo são *flags* e para os comandos *SET_ADDRESS*, *OPEN_SWITCHES*, *CLOSE_SWITCHES* e *MATRIX_WRITE*, quando estão no nível lógico '1', representam os barramentos onde deve ser enviado o comando de resposta *ACKNOWLEDGE*. Para o comando *MATRIX_READ* estes bits representam os barramentos por onde devem ser enviados os dados dos *switches*, através do comando *MATRIX_DATA*.

Capítulo 6

6. Implementação da NSU

Como já foi referido anteriormente, a NSU tem como função controlar as ligações físicas entre os nodos e os barramentos replicados. Para tal possui um dispositivo, o *switch controller*, que controla todas as operações efectuadas pela NSU, possui uma *switch table* onde está armazenada a configuração dos interruptores, uma *switch matrix* que faz a ligação física entre o(s) *transceiver(s)* do nodo e o(s) barramento(s) CAN e possui também um endereço. Estes componentes, à excepção da *switch matrix*, foram implementados através da utilização da linguagem de descrição de hardware VHDL (linguagem descrita no subcapítulo 6.3). No final deste capítulo será feita de uma forma resumida uma apresentação desta linguagem.

Neste capítulo é feita uma descrição da implementação e funcionamento dos componentes constituintes da NSU. Em anexo encontra-se uma apresentação da placa de desenvolvimento RC10 da *Celoxica* utilizada para implementar alguns destes componentes.

6.1. Arquitectura da *Switch Matrix*

A *switch matrix* é responsável pela ligação física entre o nodo e o barramento. Para possibilitar a ligação de um nodo com N=2 interfaces a M=4 barramentos CAN (ver Figura 53), foi projectado um circuito que utiliza *switches* analógicos bidireccionais. A tecnologia utilizada para implementar estes interruptores foi o interruptor analógico bidireccional *MAX4624* da *Maxim* (MAXIM, 2001). No entanto, foram consideradas outras tecnologias, tais como, as *Transmission Gate* e os *Relés Reed Switch,* mas por vários motivos que estão apresentados no apêndice A, a decisão final recaiu sobre o *MAX4624*.

Os sinais de controlo *SEL1_T1* a *SEL4_T1* e *SEL1_T2* a *SEL4_T2* representados na Figura 53 servem para seleccionar o barramento ao qual as duas interfaces vão ser ligadas e serão usados pelo *switch controller* que conforme as ordens vindas da TMU seleccionará o barramento ao qual o *transceiver* do nodo será ligado.

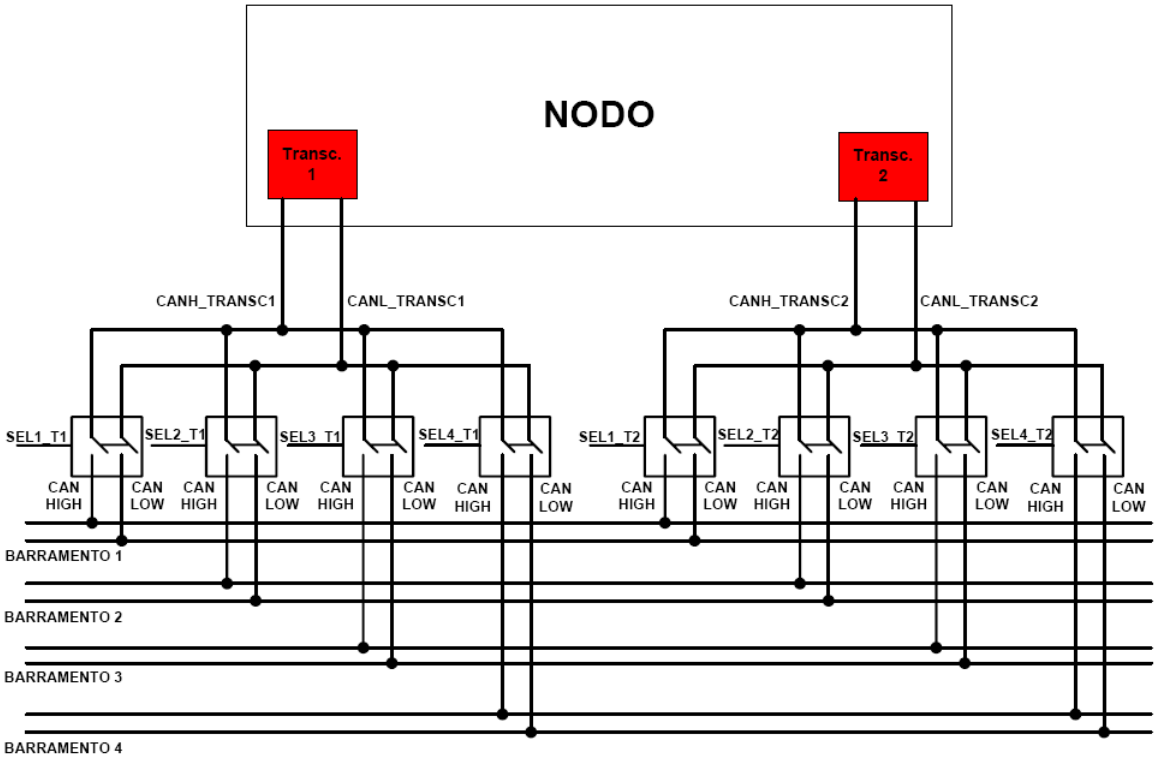


Figura 53: Exemplo com as ligações dos *transceivers* CAN do nodo aos barramentos por intermédio dos interruptores da *switch matrix* (para o caso de duas interfaces com o nodo e quatro barramentos 2:4).

A aplicação de um *multiplexer* para fazer a ligação do *transceiver* ao barramento simplificaria o circuito e até reduziria o número de circuitos integrados. No entanto, essa solução não é aplicável nesta situação por uma razão muito simples, um nodo pode estar ligado a mais do que um barramento ao mesmo tempo e assim enviar mensagens iguais em barramentos paralelos.

6.1.1. Circuito e PCB da *Switch Matrix*

De forma a possibilitar a ligação e comunicação da NSU com M=4 barramentos CAN, foi criada uma placa de circuito com 4 *transceivers SN65HVD232* da *Texas Instruments* e com os interruptores analógicos bidireccionais *MAX4624* da *MAXIM*. Esta placa foi projectada para ser ligada ao módulo de expansão da placa de desenvolvimento RC10 apresentada no apêndice A.1. No apêndice A.5 encontra-se uma descrição mais pormenorizada deste circuito.

6.2. Arquitectura do *Switch Controller*

O SC - *Switch Controller* é o componente que controla a abertura ou fecho dos interruptores da *switch* matrix e foi completamente implementado em hardware utilizando a linguagem VHDL. Na Figura 54 está apresentada a arquitectura do SC e seus componentes. O SC está normalmente num estado de espera e só começa a funcionar assim que é armazenada uma nova mensagem no *FIFO*. Enquanto o SC processa uma mensagem, novas mensagens podem surgir, sendo armazenadas no *FIFO* e processadas assim que o SC termina a sua tarefa. Este mecanismo permite evitar que sejam perdidas mensagens durante o processamento. Para gerir todos os módulos e fazer uma sincronização de todo o sistema existe uma máquina de estados.

O SC possui um sinal de relógio que é distribuído por todos os seus componentes. A frequência desse sinal de relógio é de cerca de 2Mhz, que resulta da divisão da frequência do sinal de relógio da placa de desenvolvimento RC10 que é de aproximadamente 48Mhz. Para fazer esta divisão foi implementado um divisor de relógio *CLK_Divider* que permite dividir o relógio da Placa RC10 para uma frequência mais adequada ao funcionamento da NSU. A frequência máxima a que o hardware da NSU pode funcionar pode ser obtida por consulta ao relatório

final fornecido pelo software ISE da Xilinx (Xilinx, 2008), gerado durante a sintetização do projecto. O sumário temporal foi o seguinte:

Timing Summary:

Speed Grade: -4

Minimum period: 15.675ns (*Maximum Frequency*: 63.795MHz)

O SC - *Switch Controller* é composto por vários componentes que serão descritos de uma forma detalhada nos subcapítulos seguintes. Os componentes que constituem o SC são:

- Módulo de *I/O*;
- *FIFO*;
- *MSG_Data*;
- *MSG_Control_Unit*;
- *ADDRESS*;
- *SWITCH_TABLE*;
- *Sw_Conf*;
- *MUX1*;
- *Rsp_Data*;
- *SwCtr_Ctr_Unit*;
- *CLK_Divider*.

O sumário de utilização dos recursos da FPGA da placa de desenvolvimento RC10 por parte do *Switch Controller* é o seguinte:

Número de *BUFGMUXs*	7 de 8	87%
Número de *External IOBs*	11 de 221	4%
Número de *LOCed IOBs*	11 de 11	100%
Número de *Slices*	1594 de 13312	11%
Número de *SLICEMs*	14 de 6656	1%
Número de *GCLKs*	7 de 8	87%

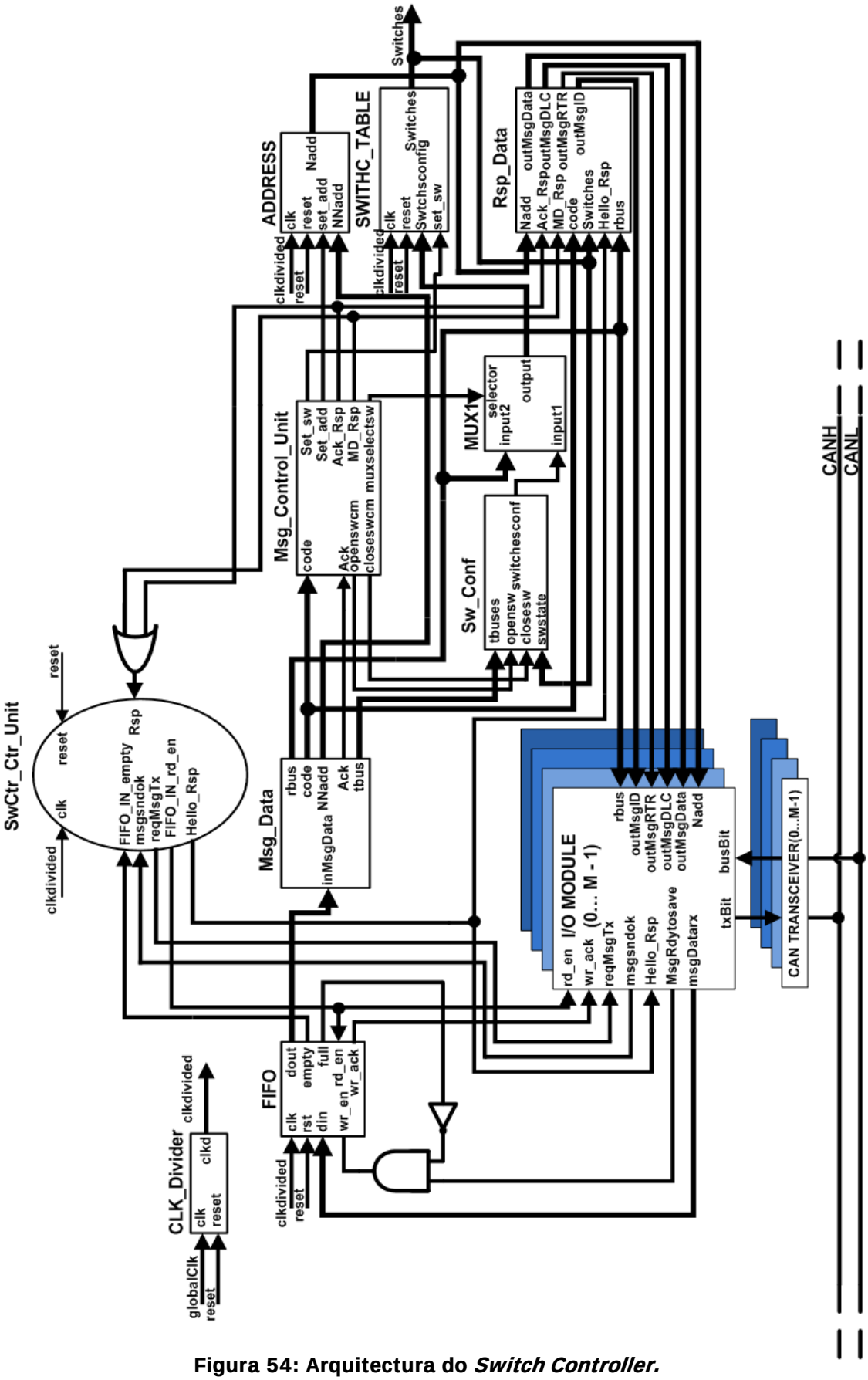


Figura 54: Arquitectura do *Switch Controller.*

Nos subcapítulos seguintes será feita uma breve descrição do funcionamento de cada um dos componentes constituintes do SC.

6.2.1. Módulo *I/O*

Este módulo permite à NSU comunicar com a TMU utilizando para isso os vários barramentos CAN e para tal possui um CLAN por cada barramento replicado (ver Figura 55). É possível replicar este módulo, uma vez que o hardware necessário para fazer a comunicação com os barramentos é igual. A linguagem VHDL facilita a tarefa de replicação deste módulo, sendo por isso muito simples adicionar ao sistema novos módulos, um por cada barramento replicado. Na Figura 56 está apresentada a arquitectura do módulo de *I/O*. Utiliza quatro controladores CLAN, um por cada barramento replicado. Estes controladores estão ligados a um mecanismo que permite armazenar mensagens num FIFO. Este mecanismo permite percorrer os quatro controladores verificando se estes possuem uma mensagem e quando é detectada uma nova mensagem num dos controladores, esta é armazenada no *FIFO*. De todo o hardware da figura os componentes *Four_Bit_Ring_Counter*, *MUX2* e *MsgTxOk_detector* são comuns em todos os módulos de *I/O*. O componente *Four_Bit_Ring_Counter* é um contador em anel e permite fazer um *scan* aos CLANs, verificando-se deste modo se foi recebida uma nova mensagem. Quando é detectada uma mensagem num dos CLANs, essa mensagem é imediatamente armazenada no FIFO que envia de seguida um sinal de confirmação (*wr_ack*). Este sinal, por sua vez, é utilizado para libertar o *buffer* de recepção de mensagens do CLAN.

O componente *MsgTxOk_detector* permite detectar quando uma mensagem foi enviada utilizando o sinal *msgTxOk* do CLAN. Quando a mensagem é enviada o *MsgTxOk_detector* envia um sinal (*msgsndok*) para a máquina de estados para que esta salte de estado. Quando a máquina de estados passa para o estado de leitura de uma nova mensagem, a *flag rd_en* é activada o que faz com que o sinal *msgsndok* seja desactivado.

O *multiplexer MUX2* permite seleccionar a informação contida no campo de dados da mensagem recebida pelo CLAN. Esses dados são armazenados no FIFO através da saída do *multiplexer* (*msgDatarx*).

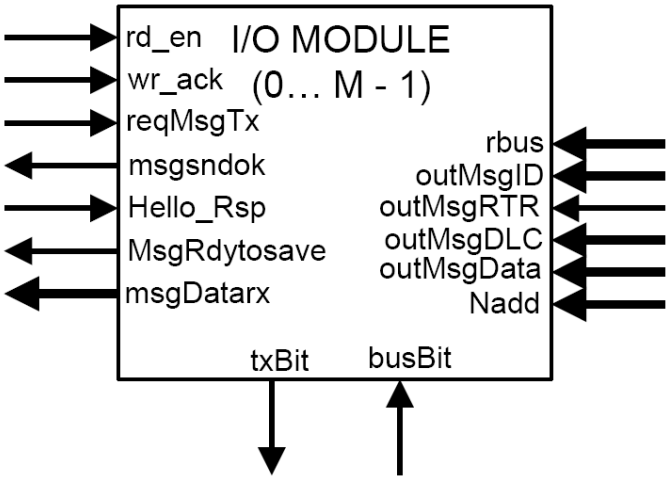


Figura 55: Módulo *I/O* da NSU.

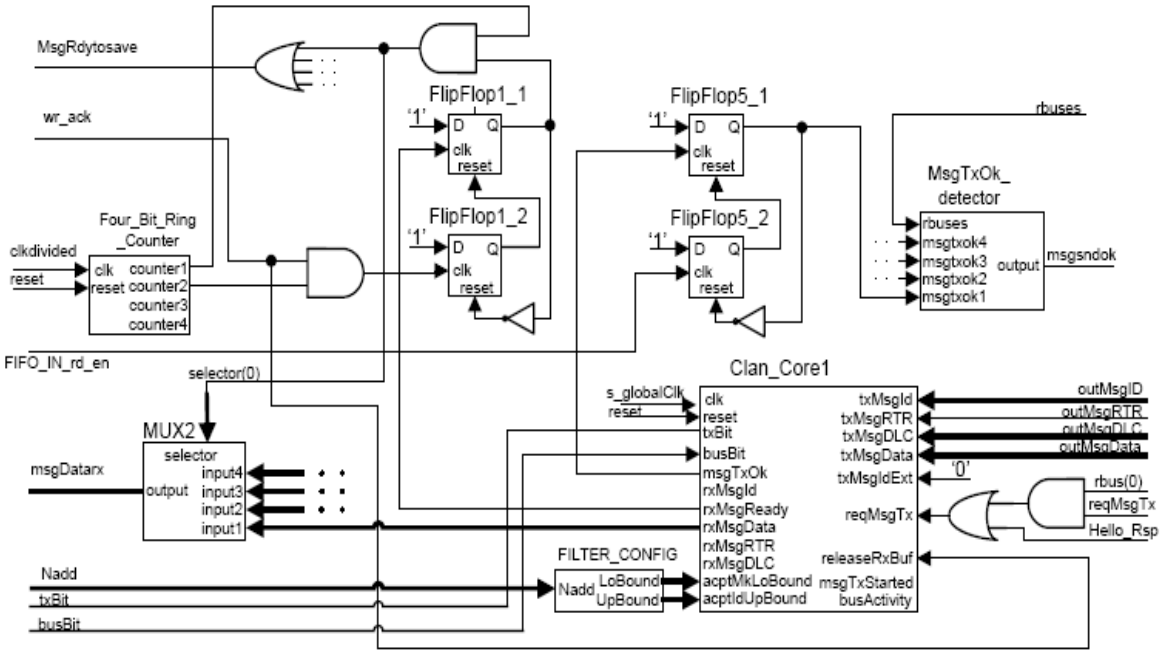


Figura 56: Arquitectura detalhada do módulo de I/O.

Na Tabela 6 estão apresentadas as portas de entrada e saída deste módulo.

Nome	Tipo	Descrição
rd_en	Entrada	*Flag* de activação da leitura do FIFO
wr_ack	Entrada	*Flag* que indica uma escrita bem sucedida no FIFO
reqMsgTx	Entrada	Quando activada, esta *flag* requer ao CLAN o envio de uma mensagem
msgsndok	Saída	Quando activada, esta *flag* indica que o envio de uma mensagem foi feito com sucesso
Hello_Rsp	Entrada	Quando activada, esta *flag* requer ao CLAN o envio de uma mensagem (neste caso especifico o comando de *Hello*)
MsgRdytosave	Saída	Quando activada, esta *flag* indica que existe uma mensagem nos *buffers* do CLAN pronta a ser armazenada no FIFO
msgDatarx	Saída	Campo de dados da mensagem
rbus	Entrada	Indica os barramentos por onde deve ser enviada a mensagem
outMsgId	Entrada	Identificador da mensagem a ser enviada
outMsgRTR	Entrada	*Flag* RTR da mensagem a ser enviada
outMsgDLC	Entrada	Valor do DLC da mensagem a ser enviada
outMsgData	Entrada	Campo de dados da mensagem a ser enviada
Nadd	Entrada	
txBit	Entrada	Nível actual aplicado no *transceiver* de saída
busBit	Entrada	Nível actual do barramento detectado pelo *transceiver* de entrada

Tabela 6:Descrição da função das portas de entrada e saída do módulo *I/O* da NSU.

6.2.2. FIFO

O FIFO foi criado através da ferramenta *Fifo Generator* da *Xilinx*. Possui uma entrada e uma saída de dados *DIN* e *DOUT* de 64 bits (ver Figura 57), sendo este o tamanho máximo do campo de dados de uma mensagem CAN. Possui duas entradas, uma para fazer a escrita e outro para a leitura, *WR_EN* e *RD_EN* respectivamente. Possui também um sinal, *FULL*, que indica quando o FIFO está cheio. O sinal *EMPTY* indica que o FIFO está vazio e o sinal *WR_ACK* é activado

quando uma nova mensagem acaba de ser escrita na memória. A capacidade de armazenamento pode ir de 16 até 4194304 palavras de 64 bits, sendo 16 palavras suficientes para implementar este sistema.

Figura 57: FIFO da NSU.

Na Tabela 7 estão apresentadas as portas de entrada e saída do FIFO.

Nome	Tipo	Descrição
clk	Entrada	Entrada de relógio
rst	Entrada	Reset assíncrono
din	Entrada	Entrada de dados
wr_en	Saída	Quando activada, esta *flag* indica que é possível fazer uma escrita
dout	Saída	Saída de dados
empty	Saída	Quando activada, esta *flag* indica que o FIFO está vazio
full	Saída	Quando activada, esta *flag* indica que o FIFO está vazio
rd_en	Entrada	Activa-se quando se pretende fazer uma leitura
wr_ack	Saída	Quando activada, esta *flag* indica que foi feita uma escrita com sucesso

Tabela 7:Descrição da função das portas de entrada e saída do FIFO.

6.2.3. *MSG_Data*

Este componente permite decompor o campo de dados de uma mensagem CAN nos vários campos que compõem o comando correspondente. Como já foi documentado na subsecção 5.5.3, os comandos são constituídos por vários campos, sendo o número de campos e o seu conteúdo variável de comando para comando. O campo de dados da mensagem CAN, fornecido pela saída *dout* do

FIFO está ligado à entrada *inMsgData* deste componente (ver Figura 58). Os campos do comando ficam disponíveis nas saídas: *rbus*, *tbus*, *code*, *NNadd* e *Ack*. Nem todas as mensagens possuem todos os campos disponíveis, no entanto, a NSU decide que campos utilizar ao verificar o tipo de comando através da verificação do campo *code*.

Na Tabela 8 estão apresentadas as portas de entrada e saída deste componente.

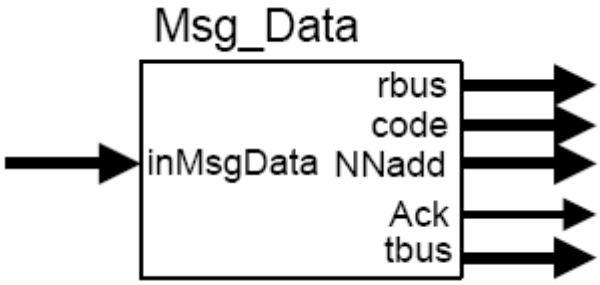


Figura 58: Componente *Msg_Data* da NSU.

Nome	Tipo	Descrição
inMsgData	Entrada	Entrada de dados (campo de dados da mensagem CAN)
rbus	Saída	Campo *rbus* do comando (*flags* indicadoras dos barramentos onde deve ser enviada a resposta)
code	Saída	Campo *code* do comando (código do comando)
NNadd	Saída	Campo *NNadd* do comando (novo endereço da NSU)
Ack	Saída	Campo *Ack* do comando
tbus	Saída	Campo *tbus* do comando (*flags* indicadoras dos barramentos alvo)

Tabela 8: Descrição da função das portas de entrada e saída do componente *Msg_Data*.

6.2.4. *Msg_Control_Unit*

O componente *Msg_Control_Unit* toma decisões do que deve ser feito dependendo do tipo de comando. Ou seja, se o comando for *SET_ADDRESS* então este componente faz com que os dados disponíveis em *NNadd* sejam armazenados no componente *ADDRESS*. Esta operação faz com que seja atribuído um novo endereço à NSU. O mesmo acontece quando se pretende alterar o estado dos *switches*, os dados disponíveis em *tbus* são armazenados na *SWITCH_TABLE*. Para isso são utilizadas duas *flags*, a *flag Set_sw* e a *flag Set_add* (ver Figura 59). Possui também as *flags*, *Ack_Rsp* e *MD_Rsp* que permitem distinguir entre o comando de *acknowledge* e o comando de resposta que envia os dados dos *switches* para a TMU. Possui mais duas *flags*, que quando são activadas podem fechar ou abrir os *switches*, estas *flags* são a *closeswcm* e *openswcm* respectivamente.

Na Tabela 9 estão apresentadas as portas de entrada e saída deste componente.

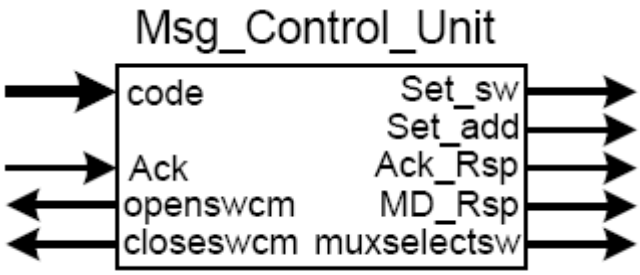


Figura 59: Componente *Msg_Control_Unit* da NSU.

Nome	Tipo	Descrição
code	Entrada	Campo *code* do comando (código do comando)
Ack	Entrada	Campo *Ack* do comando
openswcm	Saída	Quando activada, esta *flag* requer a abertura dos *switches* alvo
closeswcm	Saída	Quando activada, esta *flag* requer o fecho dos *switches* alvo
Set_sw	Saída	Quando activada, esta *flag* permite que se escreva na configuração dos *switches*
Set_add	Saída	Quando activada, esta *flag* permite que se escreva um novo endereço para a NSU
Ack_Rsp	Saída	Quando activada, esta *flag* indica que deve ser enviada uma resposta de *Ack* para a TMU
Md_Rsp	Saída	Quando activada, esta *flag* indica que deve ser enviada uma resposta com os dados da matriz de *switches* para a TMU
muxselectsw	Saída	Esta *flag* permite seleccionar a fonte dos dados para a configuração dos *switches*

Tabela 9: Descrição da função das portas de entrada e saída do componente *Msg_Control_Unit*.

6.2.5. *ADDRESS*

Este componente memoriza o endereço da NSU. Quando se pretende alterar o valor do endereço, que por defeito está definido como "000001", basta activar a *flag set_add* (ver Figura 60). É possível apagar esta memória activando a *flag reset*. Este componente é síncrono e funciona à frequência fornecida pelo sinal de relógio disponível em *clkdivided*. Em termos de hardware esta memória é simplesmente um vector de 6 bits.

Na Tabela 10 estão apresentadas as portas de entrada e saída deste componente.

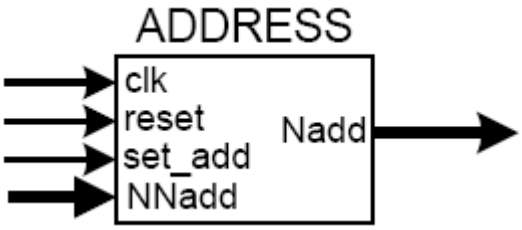


Figura 60: Componente *ADDRESS* da NSU (endereço da NSU).

Nome	Tipo	Descrição
clk	Entrada	Entrada de relógio
reset	Entrada	Reset assíncrono
set_add	Entrada	Quando activada, esta *flag* permite que se escreva um novo endereço para a NSU
NNadd	Entrada	Entrada para um novo endereço da NSU
Nadd	Saída	Endereço actual da NSU

Tabela 10: Descrição da função das portas de entrada e saída do componente *ADDRESS*.

6.2.6. *SWITCH_TABLE*

Tal como o componente *ADDRESS*, a *SWITCH_TABLE* é também um vector, mas neste caso de 4 bits. O valor definido por defeito é "1111", ou seja, todos os *switches* estão fechados. Possui também um sinal de *reset* e uma *flag set_sw* que permite configurar os *switches* quando este é activado (ver Figura 61). Funciona de forma síncrona com o relógio *clkdivided*.

Figura 61: Componente *SWITCH_TABLE* da NSU.

Na Tabela 11 estão apresentadas as portas de entrada e saída deste componente.

Nome	Tipo	Descrição
clk	Entrada	Entrada de relógio
reset	Entrada	Reset assíncrono
Swtchsconfig	Entrada	Entrada para a nova configuração dos *switches*
set_sw	Entrada	Quando activada, esta *flag* permite que se escreva uma nova configuração dos *switches*
Switches	Saída	Configuração actual dos *switches*

Tabela 11: Descrição da função das portas de entrada e saída do componente *SWITCH_TABLE*.

6.2.7. *Sw_Conf*

O *Sw_Conf* foi concebido para permitir abrir ou fechar os *switches*, utilizando informação da configuração actual, ou seja, se a operação pretendida é a da abertura dos *switches*, então o *Sw_Conf* verifica primeiro se já existem *switches* abertos e abre somente os que estão fechados. O mesmo acontece para a operação inversa (ver Figura 62).

Quando a operação pretendida é a alteração da configuração dos *switches*, por intermédio da operação *MATRIX_WRITE*, então o *Sw_Conf* verifica a configuração actual dos *switches* e altera somente o estado dos *switches* que estão num estado diferente do pretendido.

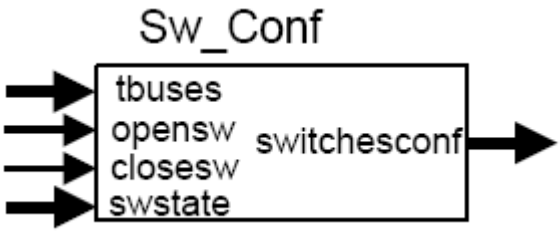


Figura 62: Componente *Sw_Conf* da NSU.

Na Tabela 12 estão apresentadas as portas de entrada e saída deste componente.

Nome	Tipo	Descrição
tbuses	Entrada	Indica os *switches* (barramentos alvo)
opensw	Entrada	Quando activada, esta *flag* requer a abertura dos *switches* alvo
closesw	Entrada	Quando activada, esta *flag* requer o fecho dos *switches* alvo
swstate	Entrada	Estado actual dos *switches*
switchesconf	Saída	Nova configuração para os *switches*

Tabela 12: Descrição da função das portas de entrada e saída do componente *Sw_Conf*.

6.2.8. *MUX1*

O *multiplexer MUX1* permite seleccionar a fonte para a configuração dos *switches* (ver Figura 63). Quando o comando é *MATRIX_WRITE* a *Msg_Control_Unit* selecciona a entrada *input2*, quando a mensagem é *OPEN_SWITCHES* ou *CLOSE_SWITCHES* então a entrada seleccionada é a *input1*.

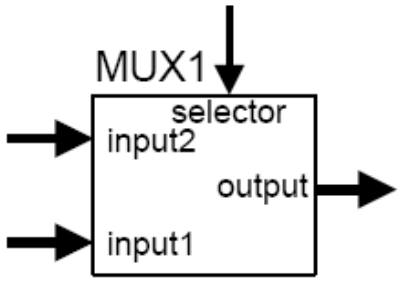


Figura 63: Componente *MUX1* da NSU.

Na Tabela 13 estão apresentadas as portas de entrada e saída deste componente.

Nome	Tipo	Descrição
input1	Entrada	Entrada de dados 1
input2	Entrada	Entrada de dados 2
output	Saida	Saída de dados
selector	Entrada	Selecciona a entrada a ser ligada à saída

Tabela 13: Descrição da função das portas de entrada e saída do componente *MUX1.*

6.2.9. *Rsp_Data*

O *Rsp_Data* é responsável pelo envio dos campos constituintes de uma mensagem, o *ID*, o *DLC*, o *RTR* e o campo de dados *DATA*, para os buffers dos CLANs (ver Figura 64). A mensagem é posteriormente enviada para a TMU. Este componente é também responsável pela distinção do tipo de mensagem a ser enviada. Faz essa distinção depois de analisar o campo *code*.

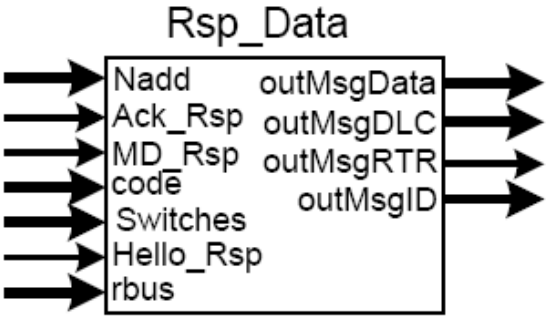


Figura 64: Componente *Rsp_Data* da NSU.

Na Tabela 14 estão apresentadas as portas de entrada e saída deste componente.

Nome	Tipo	Descrição
Nadd	Entrada	Entrada do valor actual do endereço da NSU
Ack_Rsp	Entrada	*Flag* para indicar o tipo de resposta *Ack*
MD_Rsp	Entrada	*Flag* para indicar o tipo de resposta MD (*matrix data*)
code	Entrada	Código do comando
Switches	Entrada	Configuração dos *switches*
Hello_Rsp	Entrada	Quando activada, esta *flag* indica que deve ser enviado o comando de *Hello*
rbus	Entrada	Indica os barramentos por onde deve ser enviada a resposta
outMsgData	Saída	Campo de dados da mensagem
outMsgDLC	Saída	Campo DLC da mensagem
outMsgRTR	Saída	Campo RTR da mensagem
outMsgID	Saída	Campo ID da mensagem

Tabela 14: Descrição da função das portas de entrada e saída do componente *Rsp_Data*.

6.2.10. *SwCtr_Ctr_Unit*

A *SwCtr_Ctr_Unit* é uma máquina de estados finita de *Mealy* que sincroniza todo o circuito e toma decisões do que deve ser feito dependendo do tipo de comando (ver Figura 65). É composta por quatro estados, nomeadamente o *st1_start*, o *st2_sndhellocmd*, o *st3_readmsg* e o st4_sendmsg (ver Figura 66). O primeiro estado é incondicional e apenas coloca a '1' a *flag Hello_Resp*. Este estado pode parecer à primeira vista redundante e desnecessário mas não é porque permite colocar os dados da mensagem que "transporta" o comando *Hello_Resp* nos buffers do CLAN um ciclo de relógio antes de se activar a *flag reqMsgTx*. Esta condição tem que ser respeitada sempre que se pretende enviar uma mensagem.

No segundo estado é feito o envio do comando *Hello_Resp* através da activação da *flag reqMsgTx*. A máquina de estados permanece neste estado até receber a confirmação que a mensagem foi enviada, através da visualização da

flag msgsndok. O comando *Hello_Resp* é enviado para a TMU sempre que uma nova NSU é adicionada à rede. Ao receber este comando, a TMU toma conhecimento da presença de uma nova NSU na rede e faz de seguida a sua configuração, tanto do endereço como dos *switches*.

No terceiro estado, a NSU lê a *flag empty* (*FIFO_IN_empty*) do FIFO e se esta estiver activada então o FIFO está vazio e a máquina de estados permanece neste estado até ser recebido uma nova mensagem. Se a flag empty não estiver activada então a mensagem é lida e o comando correspondente é executado. Depois de executado o comando a máquina de estados transita para o quarto estado e se a *flag Rsp* estiver activada é feito o envio da resposta para a TMU, seja uma resposta de *acknowledge* ou a configuração actual dos *switches*. A máquina de estados permanece neste estado até que seja recebida a confirmação do envio através da *flag msgsndok*. Mais uma vez, este estado pode parecer redundante mas não é pelas mesmas razões do primeiro estado, ou seja, ao mesmo tempo que o comando é executado é colocado nos buffers do CLAN os dados da mensagem de resposta que só é enviada no ciclo de relógio seguinte. Finalmente, a máquina de estados volta a ler o FIFO retomando assim o processo. A mensagem de *hello* é enviada apenas aquando da activação da NSU.

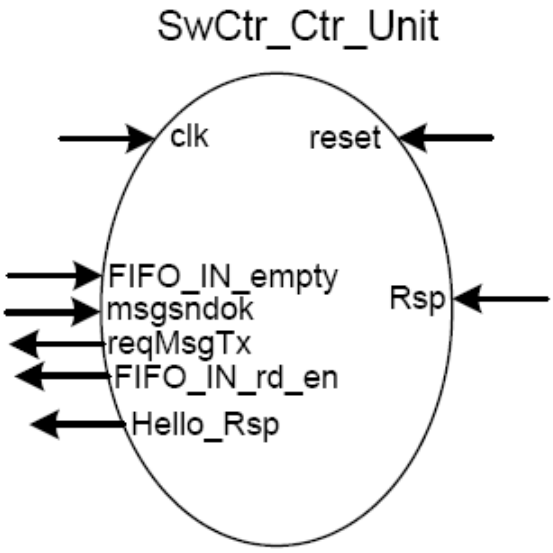


Figura 65: Componente *SwCtr_Ctr_Unit* da NSU.

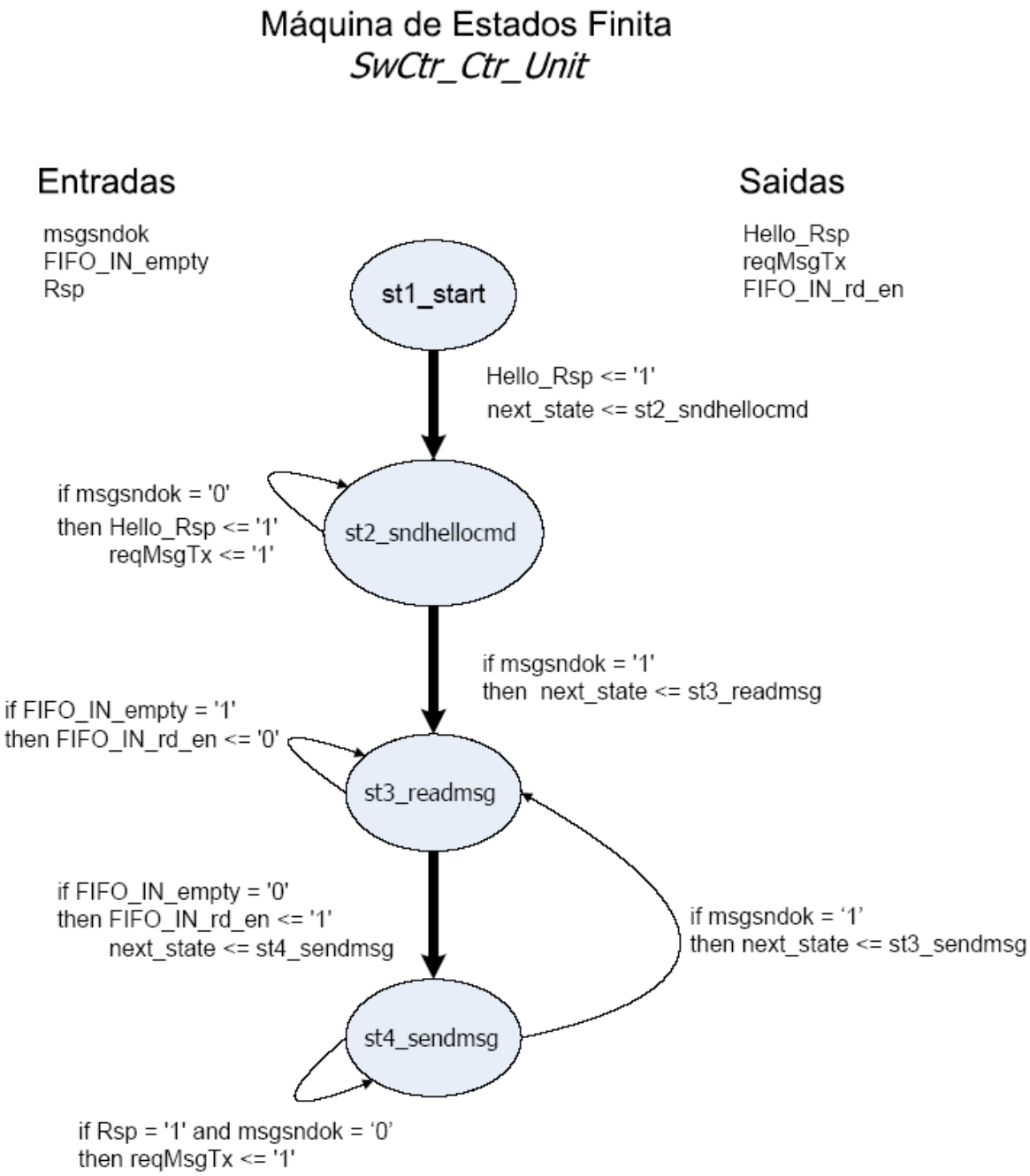


Figura 66: Diagrama da máquina de estados finita *SwCtr_Ctr_Unit*.

Na Tabela 15 estão apresentadas as portas de entrada e saída deste componente.

Nome	Tipo	Descrição
clk	Entrada	Entrada de relógio
reset	Entrada	Reset assíncrono
FIFO_IN_empty	Entrada	Quando activada, esta *flag* indica que o FIFO está vazio
msgsndok	Entrada	Quando activada, esta *flag* indica que uma mensagem foi enviada com sucesso
reqMsgTx	Saída	Quando activada, esta *flag* requer ao CLAN o envio de uma mensagem
FIFO_IN_rd_en	Saída	Quando activada, esta *flag* requer a leitura dos dados do FIFO
Hello_Rsp	Saída	Quando activada, esta *flag* requer o envio do comando *Hello*
Rsp	Entrada	Quando activada, esta *flag* requer o envio de uma resposta

Tabela 15: Descrição da função das portas de entrada e saída da máquina de estados *SwCtr_Ctr_Unit*.

6.2.11. *CLK_Divider*

O *CLK_Divider* é um divisor de relógio que permite dividir o relógio da placa de desenvolvimento RC10 para uma frequência mais adequada ao funcionamento dos componentes da NSU (ver Figura 67). Na configuração actual a NSU está a funcionar a aproximadamente 2Mhz. A frequência máxima a que o hardware da NSU pode funcionar é de 63.795MHz. Esta informação é fornecida pelo relatório final do ISE da *Xilinx*, gerado durante a sintetização do projecto. O sumário temporal foi o seguinte:

Timing Summary:

Speed Grade: -4

Minimum period: 15.675ns (*Maximum Frequency*: 63.795MHz)

Na Tabela 16 estão apresentadas as portas de entrada e saída deste componente.

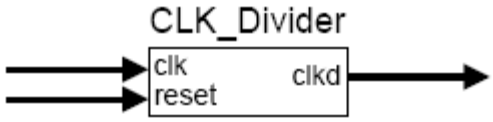


Figura 67: Componente *CLK_Divider* da NSU.

Nome	Tipo	Descrição
clk	Entrada	Entrada do relógio global
reset	Entrada	Reset assíncrono
clkd	Saída	Saída do relógio dividido

Tabela 16: Descrição da função das portas de entrada e saída do divisor de relógio *CLK_Divider*.

6.2.12. Teste Efectuado à NSU

Depois de terminado o hardware da NSU, foi realizado um teste para verificar o seu correcto funcionamento. Para este teste foi utilizada a placa de desenvolvimento RC10 (onde foi implementada a NSU) e o PCAN que permitiu a troca de mensagens entre a NSU e o *PCAN-View*. Na Figura 68 está apresentada a montagem utilizada para a realização do teste.

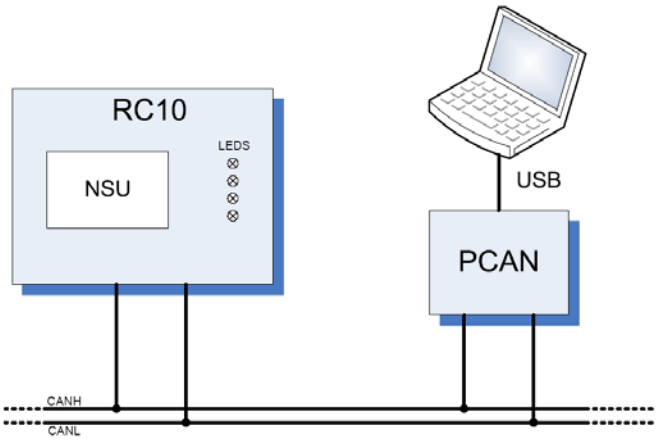


Figura 68: Montagem utilizada no teste da NSU.

Os vários comandos de configuração, descoberta e operação da NSU foram simulados e enviados através do *PCAN-View*. No apêndice C encontra-se um pequeno exemplo de configuração do *PCAN-View* para o envio de um destes comandos para a NSU. Para perceber se os comandos estavam a configurar a NSU como pretendido foram utilizados os leds presentes na placa de desenvolvimento RC10. Quando se pretendia configurar o endereço da NSU, o endereço era representado nestes leds. Quando se pretendia configurar os *switches*, era a configuração dos *switches* que aparecia representada nos leds.

O resultado do teste foi bastante satisfatório, uma vez que a NSU funcionou correctamente mesmo a 1Mbit/seg, velocidade de comunicação máxima do CAN.

6.3. VHDL (linguagem de descrição de hardware)

Em 1980 o governo dos Estados Unidos deu início a um programa de desenvolvimento de circuitos integrados de alta velocidade (VHSIC), de onde resultou o VHDL – *VHSIC Hardware Description Language*. O VHDL tornou-se assim uma linguagem standard, adoptada como tal pelo IEEE - *Institute of Electrical and Electronic Engineers* para a descrição de estruturas e funcionamento de circuitos integrados.

O VHDL é uma linguagem que permite descrever estruturas lógicas e sistemas digitais em vários níveis de abstracção, desde ao nível do sistema até ao nível de uma porta lógica. Através da utilização de entidades (*entity*) é possível descrever o funcionamento de componentes ou blocos que podem ser interligados entre si, através de sinais, para formar a estrutura de um sistema. Estas entidades possuem entradas e saídas (às quais se dá o nome de portos). A descrição da funcionalidade de uma entidade é feita através de uma arquitectura (*architecture*) podendo esta descrição ser feita de forma comportamental usando construções semelhantes às existentes nas linguagens de programação usuais, ou de forma

estrutural através da instanciação de componentes predefinidos ou na forma de fluxo de dados através da utilização de registos e barramentos. Uma combinação destes três métodos é também possível.

O VHDL possibilita a execução de tarefas ao mesmo tempo através de processos. Estes processos possuem uma lista sensitiva onde é possível definir os sinais aos quais o processo é sensível, por exemplo, o sinal de relógio ou *reset*. Um processo é activado por um evento declarado na sua lista sensitiva, por exemplo o flanco ascendente ou descendente de um sinal de relógio e executa a sua tarefa em tempo zero. Para armazenar valores numéricos ou outra informação o VHDL possui os sinais e as variáveis. Os sinais diferem das variáveis porque têm uma componente temporal associada.

O VHDL permite simular e testar os sistemas antes de estes serem fabricados, sendo possível comparar alternativas sem que isso resulte em grandes despesas.

Para conhecer o VHDL com grande detalhe é aconselhada a leitura do manual de referência *IEEE Standard VHDL Language Reference Manual*. Este documento pode ser de difícil leitura, tendo por isso sido criados vários manuais de fácil leitura, como por exemplo, o (Ashenden, 1990) ou (Ashenden, 1995).

Capítulo 7

7. Implementação da TMU

Inicialmente a TMU começou por ser projectada em hardware descrito na linguagem VHDL, como aconteceu com a NSU. Desde muito cedo se percebeu que esta tarefa seria muito complicada uma vez que a TMU é mais complexa a nível estrutural e funcional. Para facilitar a implementação da TMU pensou-se na utilização de um microprocessador, mais propriamente o *PicoBlaze* da *Xilinx* (Xilinx, 2005). A utilização deste microprocessador torna possível a implementação da TMU na linguagem de programação *assembly*, sendo assim muito reduzida a complexidade do hardware. A interface entre o CLAN e o *PicoBlaze* é feita por intermédio de um controlador. De seguida será feita uma pequena introdução ao microprocessador *PicoBlaze* e ao controlador do CLAN. É feita também uma descrição do teste efectuado à abordagem híbrida.

7.1. Microprocessador *Picoblaze*

O *PicoBlaze* é um microprocessador de 8 bits, desenvolvido pela *Xilinx* e optimizado para as FPGAs *Spartan* 3, *Virtex* II e *Virtex* II *Pro*. Este microprocessador ocupa muito pouco espaço numa FPGA, ocupa por exemplo 0.3% de uma FPGA XC3S5000. Num *block* RAM é possível armazenar 1024 instruções de programa, que são executadas desde 44 a 100 milhões de instruções por segundo (MIPS), dependendo da FPGA. Este microprocessador é excelente para implementar máquinas de estado uma vez que é fácil de programar. O facto da execução dos programas ser feita de uma forma sequencial pode ser considerado uma desvantagem quando comparado com os processos que se executam de uma forma paralela nas FPGAs, mas numa FPGA a complexidade da lógica dificulta a implementação de máquinas de estado.

O *PicoBlaze* possui as seguintes características:

- Registos de uso geral de 16 bytes
- Memória de instruções de 1K, automaticamente descarregadas durante a configuração da FPGA
- ALU de 8 bits com *flags* de indicação de *carry* e zero
- RAM interna de 64 bytes
- 256 portas de entrada e 256 portas de saída
- *Stack* automática com 31 localizações de *CALL/RETURN*
- Dois ciclos de relógio por instrução
- Resposta rápida a interrupções, 5 ciclos de relógio no pior caso
- Suporta simulação por intermédio de um assembler

Na Figura 69 estão representadas as portas deste microprocessador.

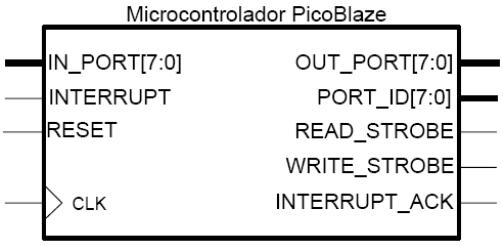


Figura 69: Portas de interface do *PicoBlaze* - adaptado de (Xilinx, 2005).

O *PicoBlaze* possui 256 portas de entrada e saída de dados, sendo cada porta de 8bits. Para ser possível identificar qual das portas está a ser utilizada, o microprocessador possui uma saída *PORT_ID[7:0]* que indica o endereço da porta de entrada ou saída. Para sinalizar um ciclo de escrita ou de leitura o *PicoBlaze* possui um sinal *WRITE_STROBE* e outro *READ_STROBE*. Quando o sinal *WRITE_STROBE* é activado, quer dizer que os dados presentes na porta *IN_PORT* foram capturados para um registo específico durante uma instrução de *INPUT*. Quando o sinal *WRITE_STROBE* é activado, quer dizer que os dados estão disponíveis na porta *OUT_PORT* durante uma instrução de OUTPUT. A saída

INTERRUPT_ACK é activado quando ocorre uma interrupção e serve para fazer o *acknowledge*.

7.1.1. Interface Controlador CLAN - *PicoBlaze*

Na Figura 70 está representado um exemplo de uma interface entre o microprocessador *PicoBlaze* e o controlador do CLAN. O *PicoBlaze* comunica com o controlador do CLAN pela porta de entrada de dados (*IN_PORT[7:0]*) e pela porta de saída de dados (*OUT_PORT[7:0]*). As saídas *WRITE_STROBE* e *READ_STROBE* permitem sinalizar um ciclo de escrita e de leitura respectivamente. A saída *PORT_ID[7:0]* indica o endereço da porta da qual os dados vão ser lidos ou escritos. Com os 8bits desta porta é possível endereçar 256 portas de entrada ou saída de dados.

A configuração, escrita e leitura dos *buffers* do CLAN "inserido" no controlador é feita por intermédio de registos de 32bits cada (consultar apêndice A.3). Cada um destes registos possui um endereço que é associado a uma porta do *PicoBlaze*. Uma vez que o barramento de dados do microprocessador é de 8bits, só se pode ler ou escrever 8bits de um registo em cada ciclo de leitura ou escrita, ou seja, são necessários quatro ciclos de leitura ou escrita para ler ou escrever num registo. Desta forma são necessárias quatro portas do *PicoBlaze* para cada registo, ou seja, cada porta corresponde a 8bits do registo. Uma vez que o controlador do CLAN possui 27 registos são necessárias 108 portas do microprocessador para permitir aceder aos 32bits de cada registo. Para endereçar as 108 portas são necessários apenas 7bits da saída *PORT_ID[7:0]* sobrado assim 1bit que pode ser utilizado para seleccionar o controlador a ser utilizado. Neste caso em particular este bit é utilizado para fazer a selecção (*chipSel*) de apenas um controlador.

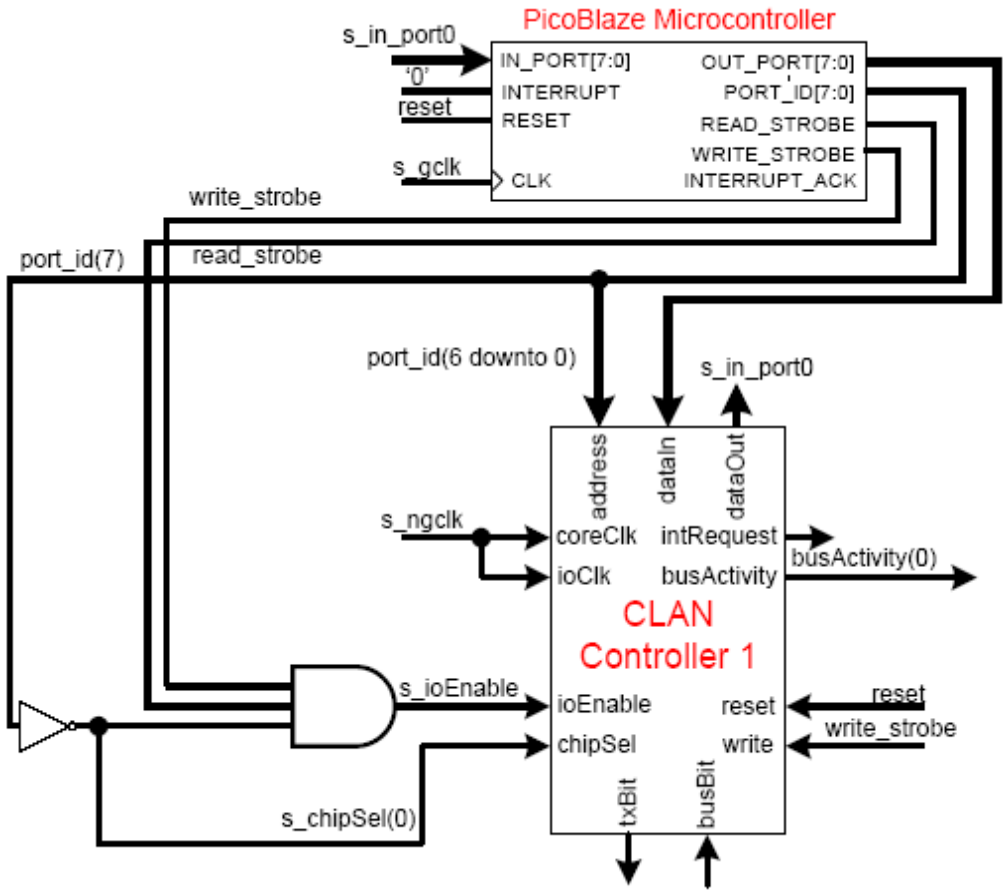


Figura 70: Exemplo da interface entre o *PicoBlaze* e o Controlador do CLAN.

7.2. Placa de Interface da TMU com os Barramentos CAN

Para possibilitar a ligação e comunicação da TMU com quatro barramentos CAN da rede, foi criada uma placa de circuito com quatro *transceivers* SN65HVD232 da *Texas Instruments*. Esta placa foi projectada para ser ligada ao módulo de expansão da FPGA. A alimentação da placa de interface é feita pela própria placa de desenvolvimento RC10 e o valor da voltagem é de +3.3V. No apêndice B.1 é feita uma descrição detalhada desta placa onde são apresentados o seu circuito e desenho da PCB.

7.3. Teste à Abordagem Híbrida Software/Hardware para a Implementação da TMU

A TMU é mais complexa a nível de funcionamento e a nível de arquitectura que a NSU, o que representa um desafio maior em termos de projecto e implementação. Para tentar minimizar esta dificuldade na implementação, foi realizado um teste para se perceber se seria possível utilizar uma solução híbrida, hardware com software (através da utilização do microprocessador *Picoblaze*), na implementação da TMU uma vez que uma implementação utilizando somente hardware se torna muito complicada e menos flexível a nível de possíveis alterações na sua arquitectura. Na Figura 71 está representado o circuito utilizado para a realização do teste a esta abordagem híbrida. O hardware utilizado consiste no microprocessador *Picoblaze*, já referido na subsecção 7.1, e consiste também no módulo de interface do CLAN já mencionado na subsecção A.3.

O software realizado, na linguagem *assembly* para o *Picoblaze*, permitiu fazer o teste em *loopback*, ou seja, uma mensagem é enviada pelo *PCAN-View* (para mais informação sobre o *PCAN-View* consultar o apêndice C) para o hardware que por sua vez a recebe e envia de volta para o *PCAN-View*. Quando o CLAN *controller* 1 recebe uma mensagem, esta é armazenada nos buffers do CLAN 1 (incluído no bloco CLAN *controller* 1) e logo de seguida é lida e armazenada na memória do *Picoblaze*. De seguida, a mesma mensagem é armazenada nos buffers do CLAN 2 (incluído no bloco CLAN *controller* 2) e enviada assim que o barramento esteja disponível de volta para o *PCAN-View*. O código desenvolvido em *assembly* para o *Picoblaze* que gere estas operações possui cerca de 300 instruções.

Este código é armazenado num *block* RAM na FPGA durante o processo de configuração da FPGA. A comunicação entre o *Picoblaze* e a memória é feita através das portas *INSTRUCTION[17:0]* e porta *ADDRESS[8:0]* (portas não representadas na Figura 71 por motivos de simplificação). Sendo a memória do

Picoblaze limitada a 1k de instruções, leva a concluir que para programas mais complexos como o de uma TMU, com todas as funcionalidades implementadas, será necessária mais memória para armazenar o código. Isto é possível utilizando para o efeito uma memória externa que comunica com o *Picoblaze* através das portas de entrada e saída de dados.

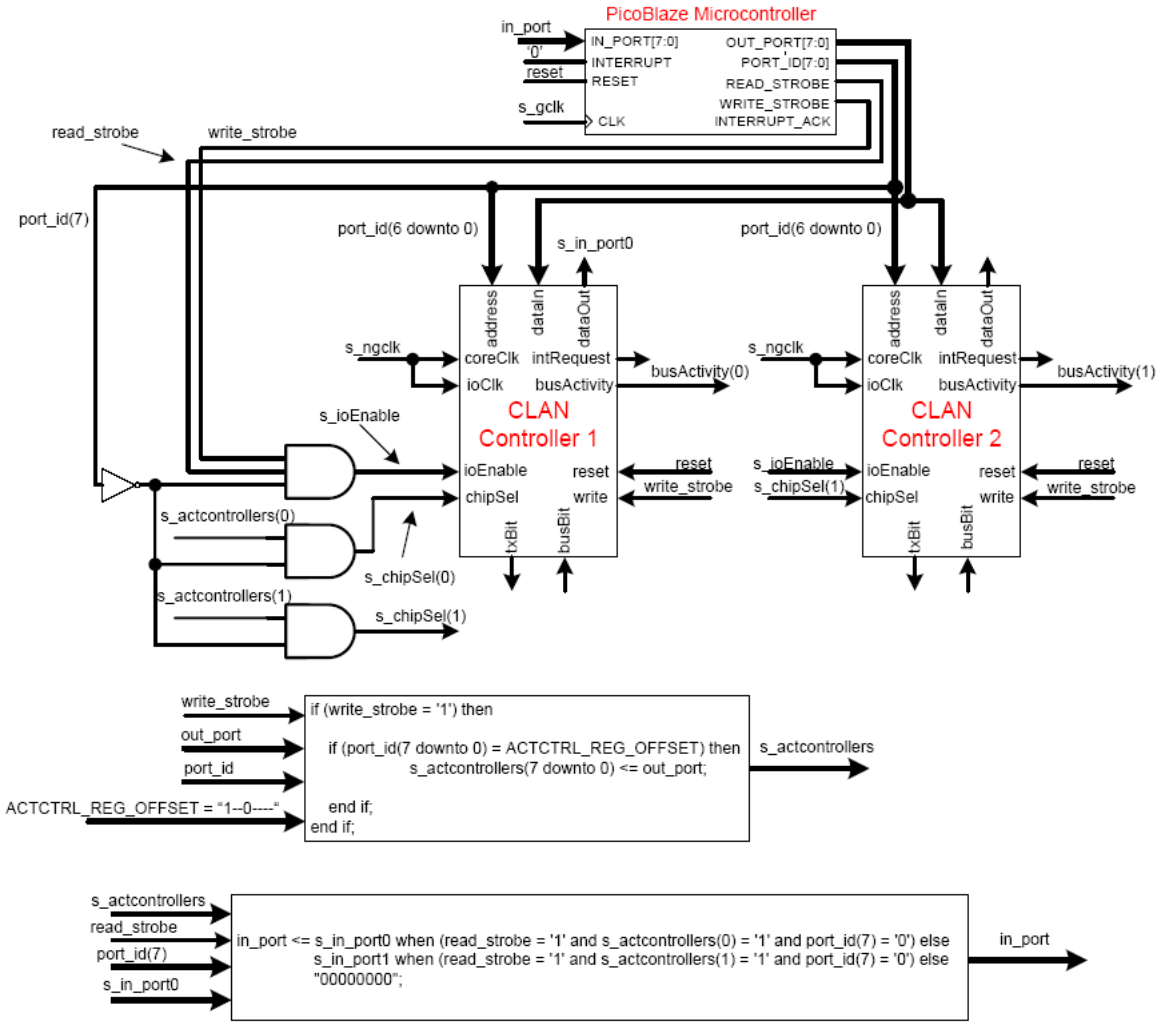


Figura 71: Circuito de teste à abordagem híbrida hardware/software.

Com este teste foi possível concluir também que, apesar da menor velocidade de processamento que esta abordagem permite em relação a uma implementação completamente em hardware, a comunicação é feita com normalidade mesmo à velocidade máxima de 1Mbit/seg.

Capítulo 8

8. Conclusão

8.1. Resumo do Trabalho Realizado

Este trabalho tem como objectivo possibilitar a introdução de barramentos redundantes e maior largura de banda a uma rede CAN. Para tal foram projectados dois tipos de componentes, uma NSU e uma TMU. A utilização de FPGAs facilitou a implementação e simulação destes componentes. Foi utilizada uma placa de desenvolvimento (a placa de desenvolvimento RC10 já mencionada na subsecção A.1) para projectar, implementar e ensaiar tanto a NSU como a TMU.

Para a implementação da NSU e da TMU foi necessário fazer um estudo detalhado do protocolo CAN e do controlador CLAN, tendo sido feito ensaios de comunicação entre o sistema em FPGA e um programa de monitorização, o PCAN apresentado no apêndice C.

A NSU foi implementada em hardware utilizando a linguagem VHDL, excepto a matriz de *switches* que foi projectada e desenvolvida num circuito de interface para a NSU que contem também os *transceivers* CAN, já mencionados no apêndice A, subsecção A.5. Esta matriz de *switches* permite ligar um nodo a vários barramentos CAN, sendo que neste caso concreto a matriz foi projectada para permitir apenas a ligação a quatro barramentos replicados. Foi realizado um teste que permitiu ajudar a determinar qual a melhor tecnologia a utilizar para implementar os *switches* da *switch matrix* (consultar subsecção A.4).

A comunicação entre a TMU e a NSU é feita através de comandos, estes comandos de operação do sistema foram mapeados em tramas CAN. Os dados relativos a um comando são transportados pelo campo de dados de uma mensagem CAN. Para simular a comunicação entre a TMU e a NSU para testar a resposta da NSU aos comandos, foi utilizado o PCAN que era configurado para

enviar mensagens cujo conteúdo do campo de dados era a informação do respectivo comando.

Para permitir o interface entre a TMU e os barramentos, foi projectado um circuito, descrito de uma forma detalhada no apêndice B.1.

Foram feitas algumas alterações ao hardware do módulo de interface do CLAN com processadores, apresentado na subsecção A.2. Essas alterações foram feitas visando facilitar a interface entre este módulo com o microprocessador *Picoblaze* de 8bits.

Durante o desenvolvimento deste projecto foram aparecendo vários desafios que foram sendo resolvidos com maior ou menor dificuldade. Sempre que necessário, foram realizados testes para ajudar a superar esses obstáculos, onde os resultados obtidos nesses mesmos testes permitiram tirar conclusões sobre o melhor caminho a tomar. Os testes realizados foram os seguintes:

- Teste a um *switch* CMOS para determinar a possibilidade da utilização desta tecnologia para implementar a *switch matrix*;
- Teste ao CLAN;
- Teste ao hardware da NSU;
- Teste para determinar a possibilidade de implementar a TMU utilizando uma abordagem híbrida software/hardware.

- Teste ao *switch* CMOS:

Este teste permitiu concluir sobre a possibilidade da utilização desta tecnologia para implementar os interruptores da *switch matrix*. Esta questão foi discutida na subsecção A.4. Foram comparadas três tecnologias, a CMOS *Transmission Gate*, os relés de lingueta e de palheta (*reed switch*) e o *switch* analógico bidireccional *MAX4624* da *MAXIM.* Concluiu-se que a melhor tecnologia a utilizar seria o *switch* analógico bidireccional *MAX4624* da *MAXIM.*

- Teste ao CLAN:

Este teste permitiu perceber como funcionava este controlador e permitiu também deixar uma estrutura (tanto a configuração como o hardware necessário) que mais tarde foi utilizada para implementar a NSU. Esta questão foi discutida na subsecção A.2.1.

- Teste ao hardware da NSU:

Depois de implementado todo o hardware da NSU foi realizado este teste para verificar se este componente respondia como esperado aos comandos de configuração. A NSU respondeu como pretendido a todos os comandos que lhe foram enviados.

- Teste à abordagem híbrida software/hardware para a implementação da TMU:

Com este teste foi possível tirar conclusões sobre a possibilidade da utilização de software em conjunto com hardware para simplificar a implementação da TMU. Esse teste permitiu tirar conclusões sobre a possibilidade de utilizar software na implementação da TMU e assim simplificar e reduzir ao máximo o hardware a utilizar. Para realizar este teste foi necessário implementar algum hardware na FPGA da placa de desenvolvimento RC10, este hardware consistiu no microprocessador *PicoBlaze* e no controlador do CLAN. O *software* realizado consistiu na linguagem *assembly* para o microprocessador *PicoBlaze*. Este software permitiu programar o *PicoBlaze* para gerir a recepção e envio de uma mensagem em *loopback*. O teste em *loopback* consistiu no envio de uma mensagem através do PCAN para o hardware na RC10, que por sua vez, recebia a mensagem e a enviava de volta para o PCAN. Concluiu-se que a abordagem híbrida pode ser utilizada sem que o desempenho da TMU seja afectado.

8.2. Direcções de Trabalho Futuro

Desta dissertação ficam por concluir alguns dos objectivos propostos, que resultam assim em pontos obrigatórios de trabalho futuro. Resultam também algumas ideias que podem ajudar a optimizar e simplificar as implementações do sistema concebido no âmbito deste mestrado integrado. Os pontos de trabalho futuro são os seguintes:

- Experimentar uma abordagem híbrida software/hardware para implementar a NSU, como a que foi pensada para a TMU. Implementar a NSU utilizando uma abordagem híbrida pode ter as suas vantagens. Uma das vantagens mais óbvias é a simplificação do hardware necessário, facilitando portanto, futuros melhoramentos e upgrades à arquitectura;
- Implementar a TMU utilizando a abordagem híbrida. Uma vez que a TMU é mais complexa a nível de funcionamento e ao nível da arquitectura que a NSU, a sua implementação em *hardware* é muito complicada. Utilizando a abordagem híbrida *software/hardware* facilita bastante a implementação;
- Implementação de um demonstrador. Um demonstrador permite de uma forma simples mostrar todo o sistema a funcionar. Seria interessante que este demonstrador pudesse introduzir falhas no sistema sempre que se desejasse verificar qual o comportamento do sistema na presença de uma falha específica, por exemplo uma partição num dos barramentos replicados.

Apêndice A

A. Informação Adicional Sobre o Hardware da NSU

A.1.Placa de Desenvolvimento RC10

A placa de desenvolvimento RC10 da *Celoxica* foi utilizada para implementar o hardware necessário à construção dos componentes NSU e TMU. Os motivos principais para a utilização desta placa de baixo custo neste projecto foram os seguintes: possui uma FPGA *Spartan* 3 da *Xilinx*, possui também um *header* de expansão de 50 pinos e um *transceiver* CAN. O *transceiver* CAN foi utilizado apenas para testar o sistema aquando da fase de projecto. Como veremos mais tarde, o *header* de expansão vai ser usado para fazer a ligação desta placa de desenvolvimento aos restantes módulos constituintes do sistema.

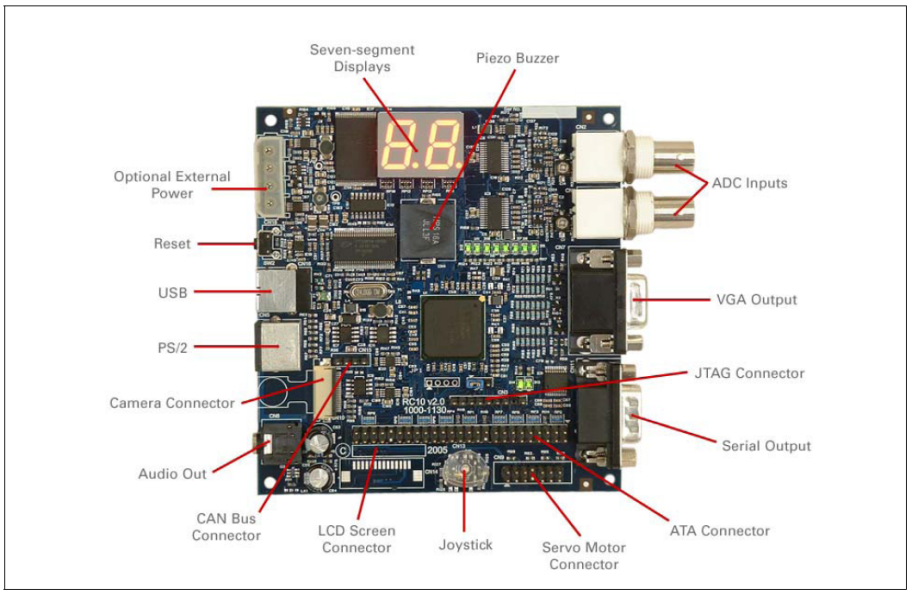


Figura A-1: Placa de desenvolvimento RC10 - retirado de (Celoxica).

A RC10 possui um relógio fixo e duas entradas para relógios externos. A ligação destes relógios aos pinos da FPGA é feita da seguinte forma:

Função	Pinos da FPGA
Relógio fixo 48.000 MHz	P10
Relógio de expansão 0	E10
Relógio de expansão 1	F10

Tabela 17: Relógios da RC10 - adaptado de (Celoxica).

Para o projecto foi utilizado o relógio fixo de 48.000 MHz. O nível lógico '1' dos sinais vindos da FPGA possui uma tensão de 3.3V com 0.5mA no máximo.

A.2.Controlador CAN (CLAN)

O CLAN é um controlador desenvolvido no Departamento de Engenharia Electrónica e Telecomunicações da Universidade de Aveiro, (Arnaldo S. R. Oliveira). Foi desenvolvido usando a linguagem de descrição de hardware VHDL e testado em FPGAs da *Xilinx*. Este controlador implementa completamente as especificações do CAN 2.0B e possui também uma interface para processadores síncrona e paralela. Possui lógica de geração de interrupções e algumas características avançadas como, filtragem de mensagens, transmissões *single shot* e estatísticas de erros. Na figura seguinte estão apresentadas todas as interfaces do controlador.

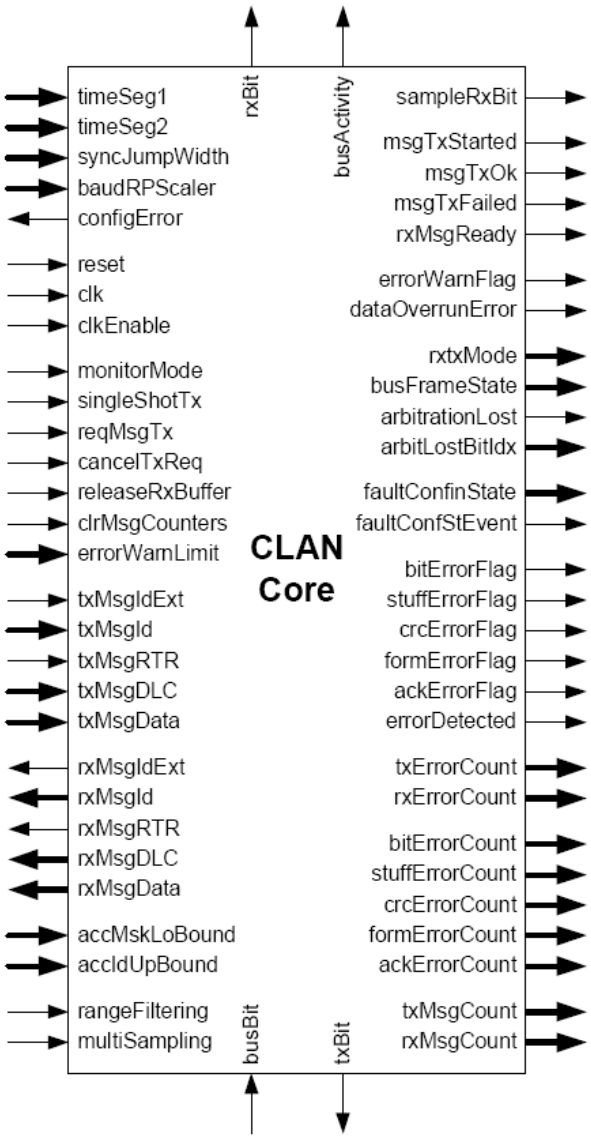


Figura 72: CLAN Core - retirado de (Arnaldo S. R. Oliveira).

O módulo de interface do CLAN com um processador está apresentado na Figura 74. O barramento de dados desta interface pode ter 8, 16 ou 32 bits de largura. O CLAN juntamente com o módulo de interface com processador ocupa cerca de 30% de uma FPGA *Xilinx Spartan-IIE XC2S300E*, o que corresponde a aproximadamente 100,000 gates lógicas.

A.2.1 Teste ao CLAN

O teste em *loopback* realizado ao CLAN serviu para verificar o correcto funcionamento deste controlador quando implementado na placa de desenvolvimento RC10. O teste consistiu na troca de mensagens entre o *PCAN-View* e o CLAN como se pode ver na Figura 73.O *PCAN-View* funciona como um nodo que envia uma mensagem CAN para o CLAN e este por sua vez recebe a mensagem que voltava a enviar de volta para o PCAN, isto a uma velocidade de transmissão de 1Mbit/seg. Na figura seguinte pode-se ver a configuração utilizada no CLAN para que este funcione em *loopback*, tal como documentado no artigo "*CLAN – A Technology Independent Synthesizable CAN Controller*" (Arnaldo S. R. Oliveira).

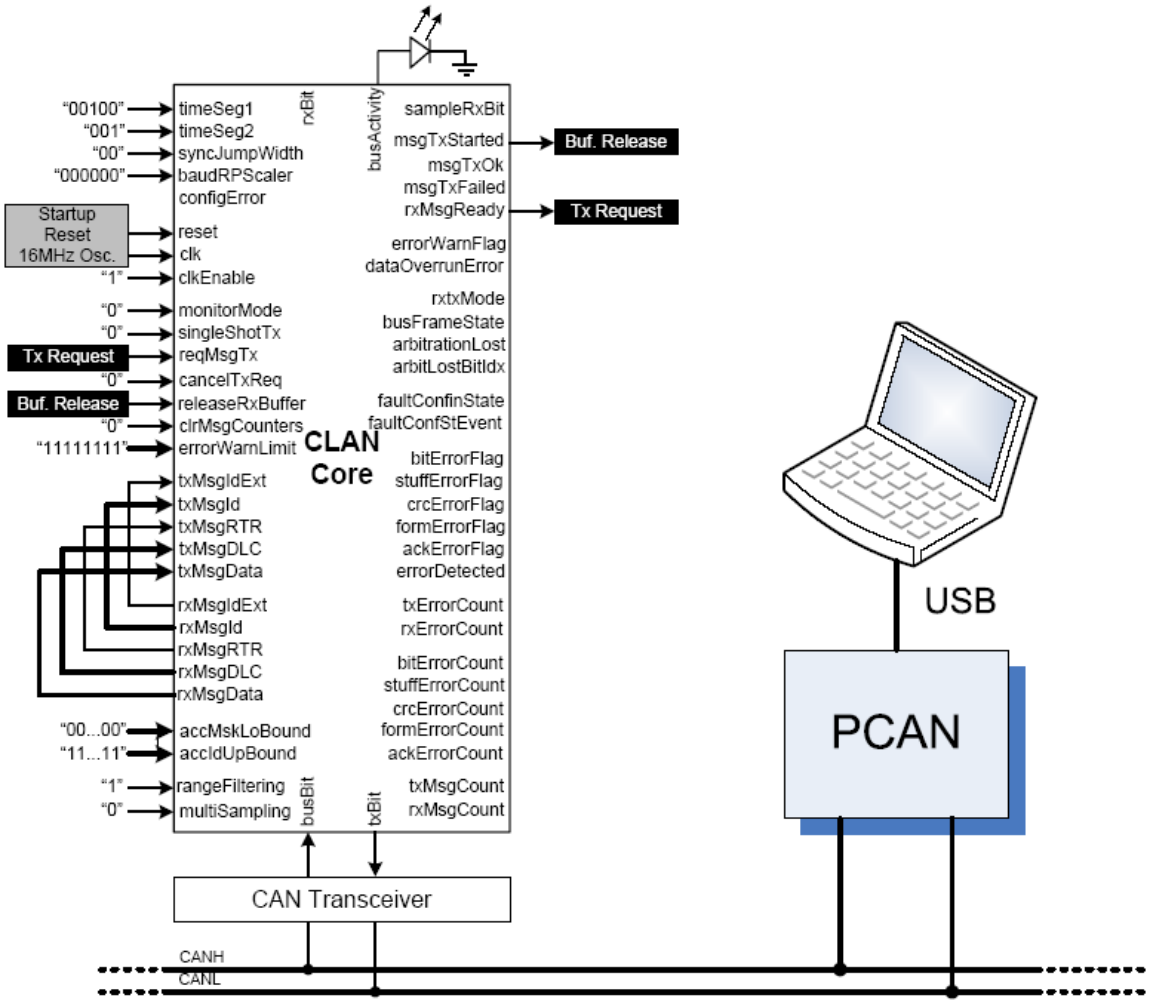


Figura 73: Montagem para o teste ao CLAN - adaptado de (Arnaldo S. R. Oliveira).

A.3. Interface CLAN com Microprocessador

Para possibilitar a interface do CLAN com microprocessadores, foi criado um módulo CLAN *Controller*, Figura 74, com um barramento de dados de 8, 16 ou 32 bits, (Arnaldo S. R. Oliveira). Nesta figura estão representadas as portas de interface deste módulo. Na Tabela 18 estão descritas as funções de cada uma das portas do módulo.

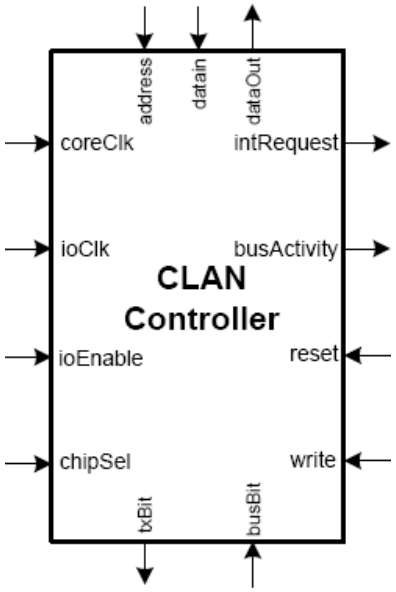


Figura 74: Módulo de interface do CLAN - adaptado de (Arnaldo S. R. Oliveira).

Este controlador possui uma série de registos que permitem configurar o CLAN que se encontra no seu "interior". Para além da configuração, estes registos permitem também ler toda a informação das mensagens recebidas e escrever novas mensagens, nos *buffers*, a ser enviadas pelo CLAN. Existem também alguns registos apenas de leitura que possuem várias *flags* que nos permitem monitorizar o CLAN mantendo-nos actualizados da ocorrência de interrupções, ou da chegada de uma nova mensagem ou até mesmo da ocorrência de erros.

Name	Type	Description
reset	*In*	Asynchronous reset input
coreClk	*In*	Core internal synchronization signal
ioClk	*In*	Interface synchronization signal
chipSel	*In*	Global interface enable signal
ioEnable	*In*	Enable signals for individual bytes of a multi-byte data bus interface
write	*In*	Write enable signal
address	*In*	Address bus
dataIn	*In*	Data input bus
dataOut	*Out*	Data output bus
intRequest	*Out*	Interrupt request for microcontroller
busActivity	*Out*	Flag that indicates activity on the bus
busBit	*In*	Port for connection to the reception transceiver (line-driver)
txBit	*Out*	Port for connection to the transmission transceiver (line-driver)

Tabela 18: Portas do módulo de interface - retirado de (Arnaldo S. R. Oliveira).

Estes registos estão mapeados em 128 endereços, tendo cada registo um máximo de 32 bits. Na Tabela 19 estão apresentados todos os registos e respectivos endereços assim como os seus acrónimos e tipo. O tipo do registo específica se é de leitura, escrita ou ambos. Na Figura 75 estão representados o ciclo de leitura e o ciclo de escrita respectivamente.

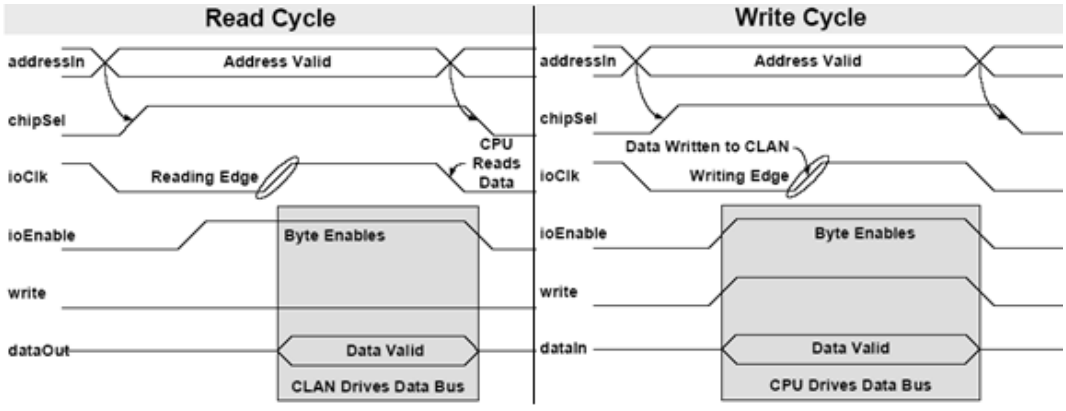


Figura 75: Ciclo de leitura e escrita do controlador - adaptado de (Arnaldo S. R. Oliveira).

Offset (Hex)	Nome	Acrónimo	Tipo
0x00	*Command*	COMDR	W
0x00	*Status 0*	STAT 0	R
0x04	*Status 1*	STAT 1	R
0x08	*Control*	CNTRL	RW
0x0C	*Rx/Tx Status*	RXTXS	R
0x10	*Bus Timing*	BUSTM	RW
0x14	*Interrupt Enable*	IENAB	RW
0x18	*Interrupt Identification*	IIDEN	R
0x1C	*Rx Error Count*	RXERC	R
0x20	*Tx Error Count*	TXERC	R
0x24	*Bit Error Count*	BITEC	R
0x28	*Stuff Error Count*	STFEC	R
0x2C	*CRC Error Count*	CRCEC	R
0x30	*Form Error Count*	FRMEC	R
0x34	*Acknowledge Error Count*	ACKEC	R
0x38	*Acceptance Mask/Lower Bound*	ACMLB	RW
0x3C	*Acceptance Identifier/Upper Bound*	ACIUB	RW
0x40	*Rx Message Count*	RMSGC	R
0x44	*Tx Message Count*	TMSGC	R
0x48	*Rx Message Control*	RXMCT	R
0x4C	*Rx Message Identifier*	RXMID	R
0x50	*Rx Message Data 03*	RXD03	R
0x54	*Rx Message Data 47*	RXD47	R
0x58	*Tx Message Control*	TXMCT	RW
0x5C	*Tx Message Identifier*	TXMID	RW
0x60	*Tx Message Data 03*	TXD03	RW
0x64	*Tx Message Data 47*	TXD47	RW

Tabela 19: Nome e endereço dos registos - adaptado de (Arnaldo S. R. Oliveira).

A.4.Tecnologias para Projecto e Implementação da *Switch Matrix*

Existem várias tecnologias passíveis de serem empregues na implementação de um interruptor (*switch*) para esta aplicação, desde que permitam fazer o *switching* de sinais analógicos. Cada tecnologia tem vantagens e desvantagens que serão analisadas e comparadas de seguida. Neste caso em particular estes interruptores têm de ser analógicos e como veremos a seguir deverão possuir algumas características especiais, por exemplo, rápidas velocidades de comutação e uma baixa resistência à passagem dos sinais.

A *switch matrix* terá que ser capaz de suportar sinais a "viajarem" a uma velocidade máxima de 1Mbit/s.

A velocidade de comutação dos interruptores varia drasticamente com a capacidade de carga que têm de "atacar". Esta capacidade por sua vez varia com o tipo de cabo, comprimento e com o número de nodos ligados ao barramento (como a *switch matrix* vai ser ligada directamente ao barramento então a capacidade diferencial Cdiff dos nodos não é considerada). Apesar de a capacidade do cabo ser bastante pequena por metro, deve ser levada em conta uma vez que o comprimento do cabo poderá atingir mil metros para velocidades de transmissão baixas. Comprimentos de cabo tão grandes implicam capacidades de carga que não podem ser desprezadas e devem ser levadas em conta aquando da escolha da tecnologia para implementar os interruptores.

- Transmission Gate:

A primeira tecnologia a ser considerada para a implementação dos interruptores foi a CMOS *Transmission Gate*. Na Figura 76 está representada uma *transmission gate* na sua forma mais simples.

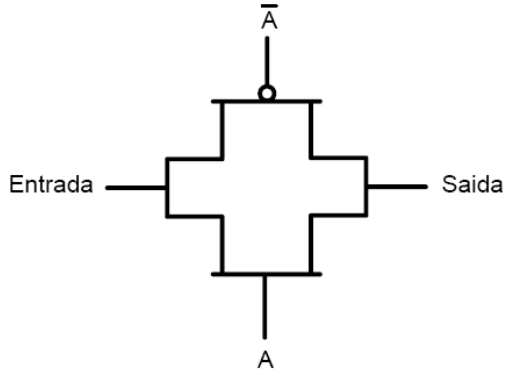


Figura 76: CMOS *Transmission Gate* - adaptado de (MAXIM, 2008).

Uma *transmission gate* é um elemento electrónico, um bom relé não mecânico construído com tecnologia CMOS. Às vezes é conhecido como porta analógica, interruptor analógico ou relé electrónico dependendo da função a que se destina. É feito através da combinação de dois transístores, um nMOS e outro pMOS em paralelo, com a entrada da gate de um transístor nMOS sendo o

complemento da entrada da gate de um transístor pMOS. As *transmission gates* são usadas para a construção de circuitos baseados em CMOS *Analog Switches*.

Modo de Operação:

Uma corrente pode fluir por este elemento em ambos os sentidos. Dependendo se existe ou não uma tensão na gate, a conexão entre a entrada e a saída pode ser de baixa ou alta impedância. Ron = 100 Ω e Roff> 5 megaΩ.

O modo de funcionamento pode ser entendido de outra forma: quando a entrada da gate do transístor nMOS está no nível lógico '0' e a entrada da gate do transístor pMOS está no nível lógico complementar '1', ambos os transístores estão desligados. No entanto, quando estamos perante o caso contrário, ou seja, a entrada da gate do transístor nMOS no estado lógico '1' e a entrada da gate do transístor pMOS no estado lógico '0', os dois transístores estão ligados ficando a gate transparente passando qualquer sinal sem degradação. O uso de gates de transmissão elimina o indesejado efeito de *threshold*.

- Relés *Reed Switch:*

Os relés de lingueta e de palheta (*reed switch or dry reed contact*) foram criados para isolar a superfície dos contactos da contaminação atmosférica, responsável pela alteração do valor das suas resistências. Nos de palheta a solução foi envolver as suas partes móveis numa atmosfera inerte, geralmente nitrogénio seco. Na figura seguinte, Figura 77, está representado numa versão muito simplificada um relé de palheta ou *Reed Switch*, montado num tubo de vidro com gás inerte para evitar a corrosão dos contactos. Quando uma corrente percorre a bobine, um campo magnético é criado e obriga os dois contactos a unirem-se fazendo assim contacto.

Este tipo de relé consegue assim comutações mais rápidas que um relé convencional e um tempo de vida superior.

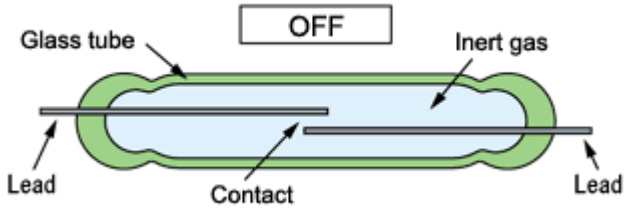


Figura 77: Relé de palheta ou Reed Switch - retirado de (Cika, 2007).

Esta tecnologia é mais interessante que as *transmission gates*, uma vez que praticamente não oferece resistência à passagem dos sinais. O tempo que demora a fazer contacto ou a libertar é bastante pequeno, da ordem dos 100µs. Como se trata dum componente electromecânico possui um tempo de vida, ou seja, um tempo de funcionamento finito da ordem do bilião de ciclos. Embora este tempo de vida seja bastante elevado, a sua localização no circuito deve ser de fácil acesso para possíveis substituições.

Existem alguns circuitos comerciais, com dimensões bastante reduzidas, o que facilita a integração nos circuitos ficando estes também com dimensões reduzidas. Devido aos materiais utilizados na sua construção é possível a sua utilização em telecomunicações com frequências de comutação na ordem dos vários GHz.

Como neste caso o relé seria ligado a um cabo que possui uma capacidade equivalente dependendo do seu comprimento, medidas devem ser tomadas para proteger o relé de forma a aumentar a sua esperança de vida e diminuir a energia gasta nas comutações entre aberto ou fechado. Como o relé possui uma resistência muito pequena, neste caso 100mΩ, a corrente instantânea que percorre este componente necessária para carregar a capacidade do cabo é muito elevada. Durante o tempo em que a capacidade é carregada o relé vai ter que dissipar uma potência enorme que pode derreter os seus contactos. Segundo (Instruments, 2006), basta ligar uma resistência em série com a capacidade equivalente para limitar esta corrente. Essa resistência Rp deverá ter um valor superior a:

$$\frac{V}{Iswitching} < Rp$$ [adaptado de (Instruments, 2006)]

Onde V é a tensão que atravessa o relé, Rp é a resistência que se deve adicionar e *Iswitching* é a corrente máxima de *switching* especificada pelo fabricante do relé. Com esta resistência o relé fica então protegido contra o aquecimento excessivo causado pela dissipação da energia, aquando o carregamento da capacidade de carga. Quanto maior for a capacidade de carga, maior é o tempo necessário para a carregar, o que provoca um aquecimento excessivo durante um tempo τ (τ=Rcontacto×Ctotal, Ctotal=Centre contactos + Ccarga). Este tempo, a uma corrente elevada, pode como já foi referido, derreter os contactos do relé.

Para o relé (MEDER, 2008), está especificada uma corrente de *switching* de 0.5A. No subcapítulo 3.3.5 vimos que a tensão máxima atingida por um sinal num barramento CAN é cerca de 3V, então, fazendo as contas chegamos a um valor para Rp de 6Ω. Este valor obtido para a resistência Rp é muito pequeno quando comparado com a resistência Ron=100Ω do *switch* na tecnologia CMOS. Na Tabela 20 estão representados os dados referentes à bobine do relé.

Contact Form	Switch Model	Coil Voltage		Coil Resistance			Pull-In Voltage	Drop-Out Voltage	Nominal coil Power
All Data at 20 °C *		VDC		Ω			VDC	VDC	mW
		Nom.	Max.	Min.	Typ.	Max.	Max.	Min.	Typ.
1A	80	5	7.5	135	150	165	3.75	0.75	167
1A	80	3	5	63	70	77	2.25	0.45	129
* the pull-in / drop-out voltages and coil resistance will change at the rate 0,4% per °C									

Tabela 20: Dados da bobine fornecidos pelo fabricante - retirado de (MEDER, 2008).

Esta tabela possui os valores da tensão DC na bobine, a sua resistência, a tensão de entrada e saída, voltagem de activação e desactivação e potência. A tensão nominal da bobine nesta tabela é de 5V mas existe uma opção de 3V para este componente. Como é óbvio a opção de 3V é mais interessante porque assim

é possível ligar este componente directamente à FPGA. É apenas necessário incluir um díodo para proteger a FPGA de possíveis correntes inversas vindas da bobine.

O preço anunciado pela RS Online para este componente é de 4,30€ por cada relé. Tendo em consideração que cada *switch* igual ao já visto na subsecção 6.1 necessita de 2 relés destes e que a *switch matrix* com N=2 interfaces e M=4 barramentos necessita de 8 *switches*, então o preço estimado para a construção do circuito será aproximadamente 68,8€. Mantendo o número de barramentos como quatro, então acrescentar mais um nível de redundância na ligação nodo barramento implica um acréscimo de 34,4€. Adicionar um barramento redundante implica um acréscimo de 8.6€ por cada ligação redundante entre o nodo e o barramento. A este valor é ainda necessário incluir o preço dos díodos, resistências e inversores necessários para inverter os sinais de controlo. O preço do componente torna inviável a sua utilização no projecto, uma vez que se pretende a solução mais económica possível.

- *Switch* analógico *MAX4624* da *Maxim:*

O *switch* analógico bidireccional *MAX4624* da *MAXIM* possui uma baixa resistência Ron e pode ser alimentado com 3.3V. Estas características aliadas às pequenas dimensões que permitem um elevado grau de integração, fazem com que este componente seja aplicável à construção da *switchmatrix*. Na figura seguinte é possível ver a configuração dos pinos e o diagrama funcional/tabela da verdade do *switch*. Como se pode ver, os pinos 4 e 6 são as saídas, sendo o pino 1 (IN) a entrada de comando que permite comutar a saída entre NO e NC. O pino 4 (NC) está, tal como o nome indica, normalmente fechado e o pino 6 (NO), está normalmente aberto. O pino 5 (COM) é o pino de entrada ou saída, dependendo do sentido do sinal (MAXIM, 2001).

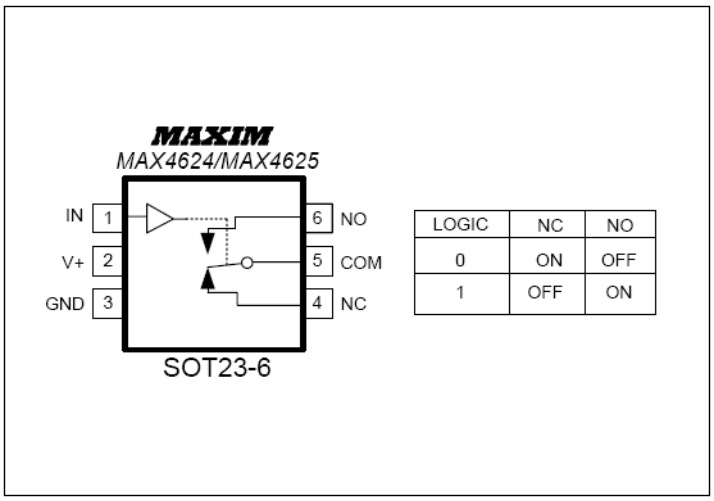


Figura 78: Vista superior do componente MAX4624/MAX4625 e sua tabela da verdade - adaptado de (MAXIM, 2001).

Na tabela seguinte é possível ver a função que cada pino tem no *switch*.

Pino	Nome	Função
1	IN	Entrada digital de controlo
2	V+	Entrada de alimentação positiva
3	GND	Massa
4	NC	Interruptor Analógico – Normalmente Fechado
5	COM	Interruptor Analógico – Comum
6	NO	Interruptor Analógico – Normalmente Aberto

Tabela 21: Nome e função de cada pino do componente - adaptado de (MAXIM, 2001).

Este *switch* é um *mux/demux*, dependendo do sentido do sinal. Esta configuração permite diminuir o número de componentes a utilizar na *switchmatrix*, em comparação com as tecnologias anteriores. O preço praticado pela RS Online para este componente é de 3,06€ a unidade, mas comprando em quantidades superiores a cem, o preço baixa para os 2,16€. Comparando o preço final da *switchmatrix* usando esta tecnologia com o da tecnologia anterior, o preço é mais reduzido, sendo neste caso cerca de 34,56€.

Escolha da tecnologia a utilizar:

De entre várias *trasmission gates* um dos circuitos com melhor desempenho é o *High-Speed* CMOS *Logic Quad Bilateral Switch* CD74HC4066 da *Texas Instruments*. Este circuito apresenta tempos de comutação bastante baixos e é possível ser alimentado com 3.3V. Foram estas as características principais que aparentemente o tornavam aplicável à realização da *switch matrix*. No entanto, este circuito possui uma característica que deriva directamente da sua tecnologia e que colocou em dúvida a possibilidade da sua utilização neste projecto. Esta característica é a sua resistência Ron, que para uma voltagem de alimentação do circuito de 3.3V é de cerca de 100Ω. Realizou-se um teste para tirar esta dúvida que surgiu aquando do projecto da *switch matrix*. Quando se liga um *switch* CMOS, que possui uma resistência Ron, ao barramento, essa resistência em conjunto com a capacidade característica por metro desse barramento, vai formar um filtro RC. A dúvida surgiu, uma vez que o filtro RC pode afectar as transições, estado dominante - recessivo e vice-versa, de tal forma que a comunicação entre os nodos deixe de ser possível.

Para a concretização deste teste foi realizado um modelo que permitiu simular a comunicação entre dois nodos estando uma resistência Ron (que simula a resistência que o *switch* oferece quando está ligado (circuito fechado)), ligada entre o nodo e o barramento. A resistência Ron=100Ω foi obtida por consulta ao *datasheet* do circuito CD74HC4066 que possui quatro *switches* analógicos na tecnologia CMOS. O barramento, por sua vez, como se trata de uma linha de transmissão bifilar, possui uma capacidade parasita por metro. Para simular o barramento não foi possível utilizar cabo real, uma vez que os recursos do laboratório não o permitiram. Comprimentos elevados de cabo, como seria necessário para realizar este teste, seriam muito dispendiosos. Para resolver este problema pensou-se em utilizar uma capacidade equivalente Ceq, igual à que um cabo real teria para a distância a ser testada. Esta capacidade equivalente foi inserida entre o CANH e o CANL do barramento. Foram utilizados vários valores de capacidades para Ceq, para os vários comprimentos do cabo a ser testado, como

se pode ver na tabela seguinte. Este Ceq foi obtido por consulta ao *datasheet* do cabo BELDEN 3105ª. Este cabo é utilizado em comunicações e automações em ambientes complicados uma vez que possui malha metálica.

Comprimento do barramento (metros)	Capacidade equivalente, Ceq (Mutual capacitance approx. 36.1 pF/m)
30	**1.1nF**
100	**3.6nF**
250	**9nF**
500	**18.1nF**

Tabela 22: Capacidade equivalente por comprimento do barramento.

O circuito utilizado na simulação está representado na figura que se segue (ver Figura 79).

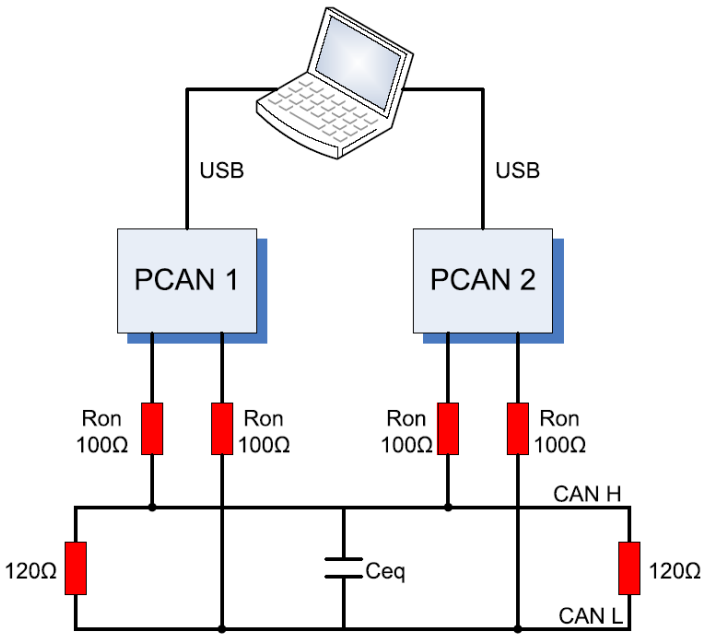


Figura 79: Montagem utilizada no teste.

Foi utilizado o programa de monitorização *PCAN-View*, descrito no apêndice C, para simular os nodos e assim se fazer a comunicação entre eles. O PCAN 1 da

figura é o nodo 1 e o PCAN 2, o nodo 2, estando estes ligados a um barramento por intermédio da resistência Ron que simula a resistência do interruptor CMOS.

O resultado deste teste não foi muito animador uma vez que a comunicação entre os nodos não se concretizou para nenhum dos valores de Ceq. Ou seja, sempre que era enviada uma mensagem pelo nodo 1, esta não era recebida no nodo 2. Esta situação verificou-se para todos os valores de Ceq. Inicialmente atribuiu-se este fracasso na comunicação ao facto de a resistência Ron ser elevada, mas um estudo mais aprofundado da camada física do CAN permitiu concluir que este insucesso se deveu ao facto de se ter introduzido a capacidade (Ceq) entre o CANH e o CANL, que é muito elevada. Como a impedância característica do barramento sem nenhuma carga ligada a ele é $Z = \sqrt{\frac{L}{C}}$, ao se ligar ao barramento uma carga com uma certa capacidade C', a impedância diminui nessa zona e passa a ser $Z' = \sqrt{\frac{L}{(C + C')}}$ (fórmula obtida por aproximação). Segundo (Steve Corrigan, Texas Instruments, 2008), a impedância Z' terá que ser superior a $0.71Z_0$ ($Z' > 0.71Z_0$), (sendo Z_0 a impedância característica do barramento), para que o valor da tensão limite para o estado recessivo de 0.5V seja respeitado. Para C' = Ceq(30metros) = 1.1nF, Z_0 = 120Ω e para C = 36.1pF temos $Z' = Z_0\sqrt{\frac{C}{(C + C')}}$ resultando em Z' = 0.178Ω. Assim sendo a condição $Z' > 0.71Z_0$ nunca é satisfeita. Não se pode simplificar o barramento reduzindo-o a uma simples capacidade equivalente, não sendo portanto, possível realizar o teste desta forma.

Uma vez que as ligações entre o nodo e o barramento (*stub*) não são terminadas, podem existir reflexões que alteram o nível dos sinais e provocam erros. Para minimizar estas reflexões, o comprimento dos *stubs* não pode exceder um terço (1/3) do comprimento crítico da linha. O comprimento crítico do barramento ocorre quando tempo de propagação do sinal pela linha iguala o tempo de transição do sinal. Para uma velocidade de comunicação de 1Mbit/seg o

comprimento recomendado para um *stub* é de 0.3m. Desta forma conseguimos garantir que o *stub* fica invisível para a linha de transmissão. Para reduzir ao máximo as reflexões, para além desta preocupação com o comprimento do *stub*, a impedância diferencial do *transceiver* deve ser a mais elevada possível. Quanto maior for essa impedância, maior é o número de nodos que se pode ligar ao barramento, uma vez que a corrente solicitada quando é representado o estado dominante no barramento, é menor. A resistência Ron do *switch* não afecta a maneira como o barramento "vê", tanto o *stub* como o *transceiver*, uma vez que esta contribui para um aumento da impedância diferencial do *transceiver*.

Este teste não permitiu tirar conclusões quanto à utilização desta tecnologia para implementar a *switch matrix*. A solução para este problema passou por procurar tecnologias com uma resistência Ron muito baixo, e assim ter a certeza que este aspecto não iria afectar os sinais de tal forma que comprometesse a comunicação entre os nodos. Das tecnologias consideradas, apenas o *switch* analógico bidireccional MAX4624 da Maxim, e o relé de palheta ou *Reed Switch* possuem uma baixa resistência Ron. Destas duas tecnologias, foi escolhido para a implementação o *switch* da Maxim, uma vez que é mais barato, mais simples e mais fiável que os relés, para mais informação sobre estes componentes devem consultar a subsecção A.4.

A.5.Circuito da *Switch Matrix*

Nas três figuras seguintes (ver Figura 80,

Figura 81 e

Figura 82) está apresentado o circuito da *Switch Matrix*. Este circuito possui, para além dos *switches* da *switch matrix*, os *transceivers* necessários para fazer a comunicação com os barramentos CAN. Possui também leds que indicam quais os barramentos com actividade e leds que indicam os sinais de activação dos *switches* que estão activos (SEL1_T1, SEL2_T2, etc.).

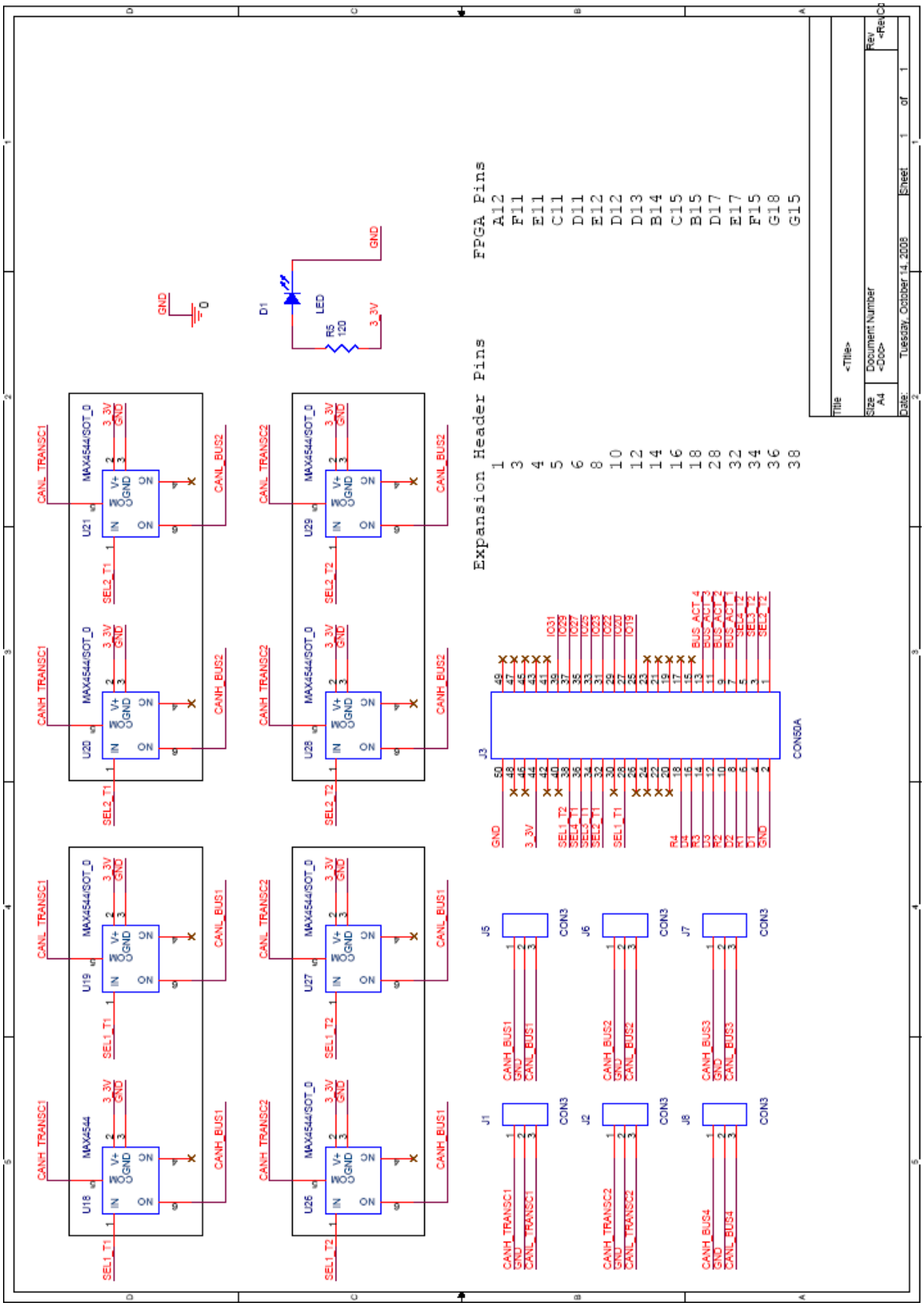

FPGA Pins
A12
F11
E11
C11
D11
E12
D12
D13
B14
C15
B15
D17
E17
F15
G18
G15
Expansion Header Pins
1
3
4
5
6
8
10
12
14
16
18
28
32
34
36
38
CON50A
CON3
MAX4544
MAX4544/SOT_0
CANH_TRANSC1
CANL_TRANSC1
CANH_TRANSC2
CANL_TRANSC2
CANH_BUS1
CANL_BUS1
CANH_BUS2
CANL_BUS2
CANH_BUS3
CANL_BUS3
CANH_BUS4
CANL_BUS4
SEL1_T1
SEL1_T2
SEL2_T1
SEL2_T2
LED
R5
120
3_3V
GND
Title
<Title>
Size
A4
Document Number
<Doc>
Rev
<RevCo
Date:
Tuesday, October 14, 2008
Sheet
1
of
1

Figura 80: Página 1 do circuito da *Switch Matrix*.

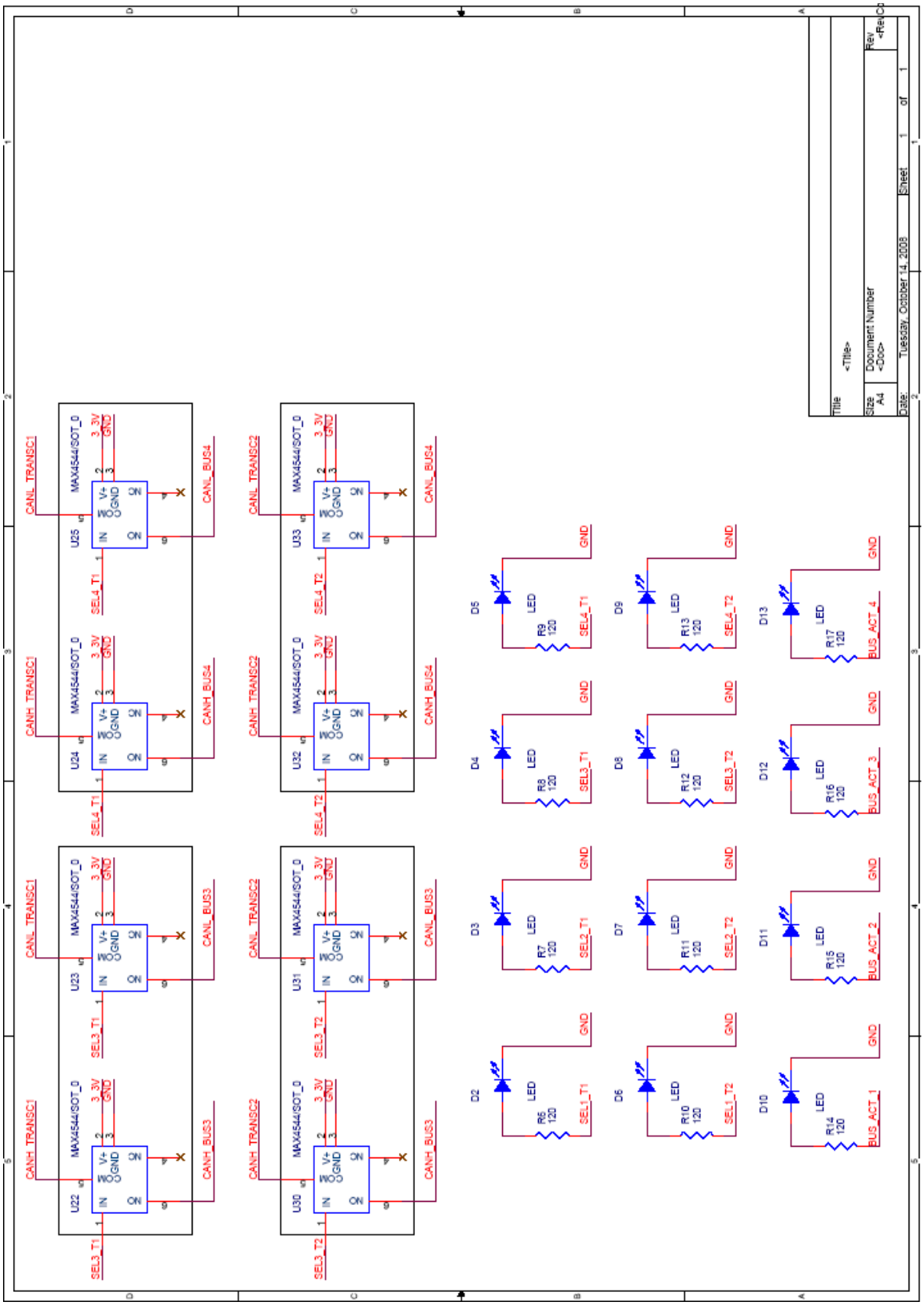


Figura 81: Página 2 do circuito da *Switch Matrix*.

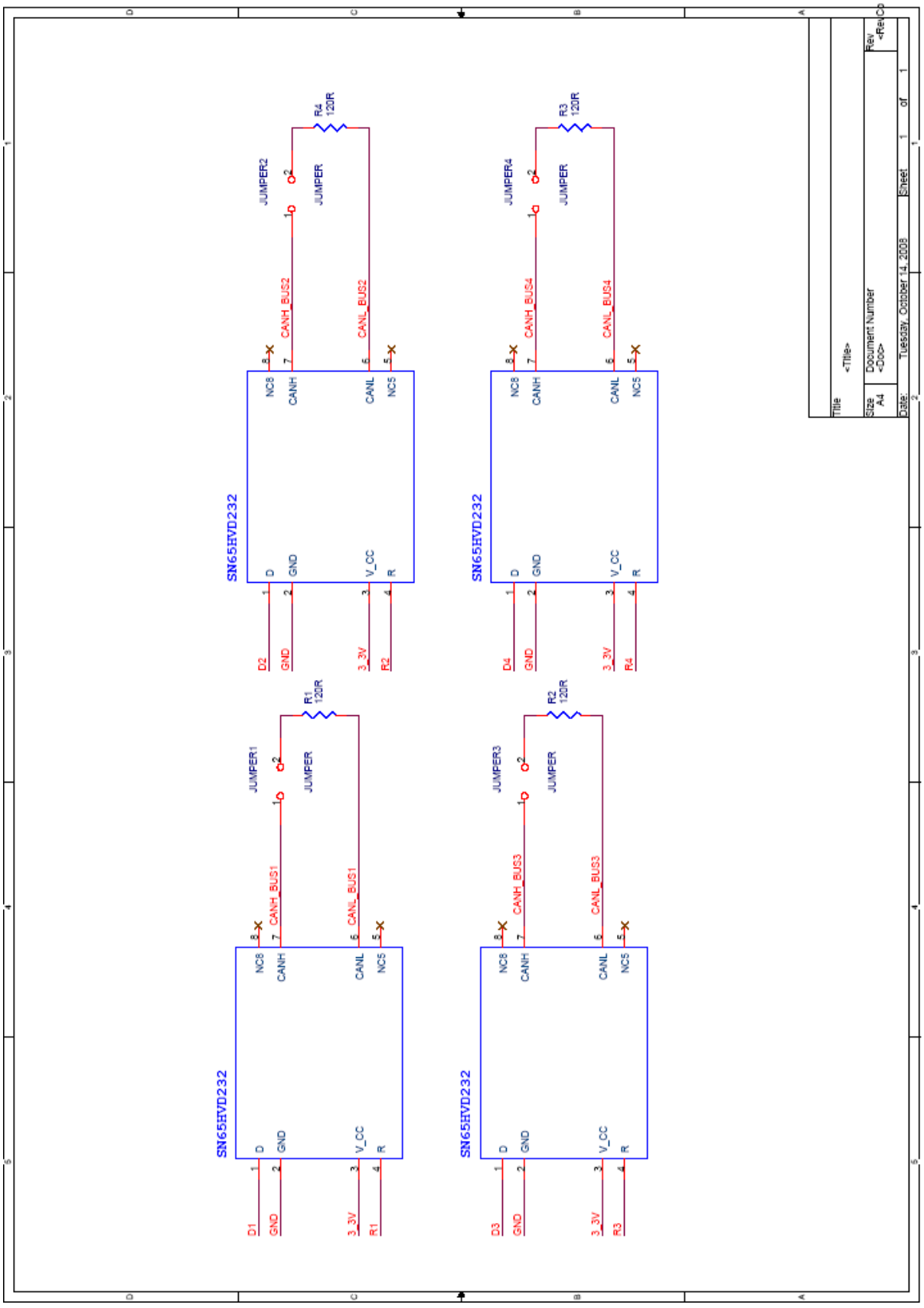


Figura 82: Página 3 do circuito *Switch Matrix*.

As placas de circuito impresso do circuito da *switch matrix* encontram-se nas figuras seguintes. Estas placas de circuito impresso foram desenhadas com a ferramenta *Orcad Layout* e a sua forma e dimensões foram escolhidas para que a PCB "encaixe" na placa RC10, mencionada na subsecção A.1.

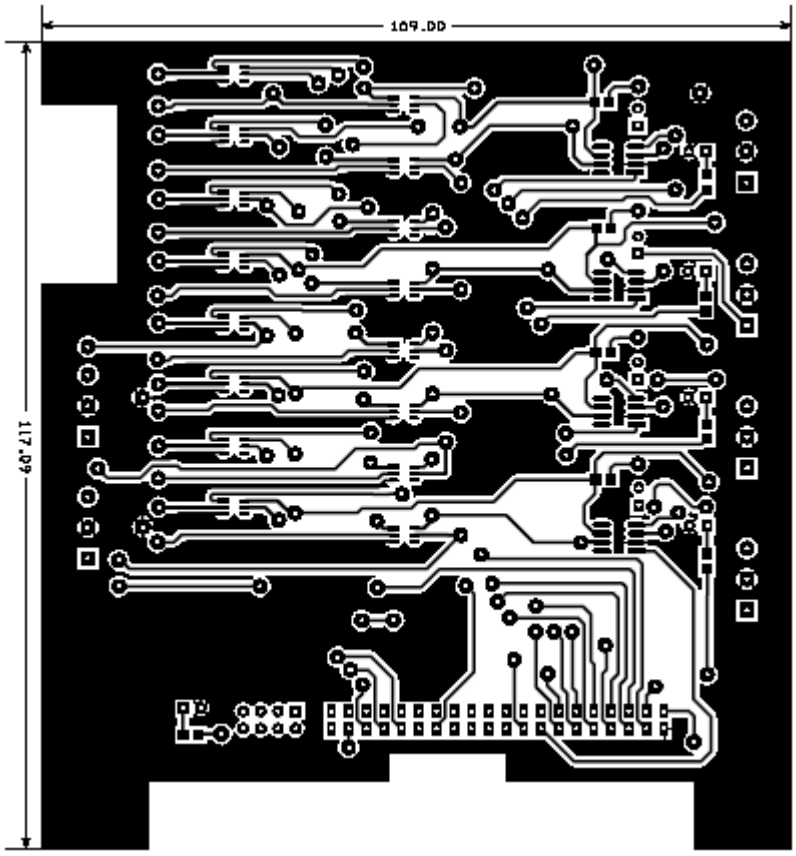


Figura 83: Face de "cima" da PCB (dimensões em milímetros).

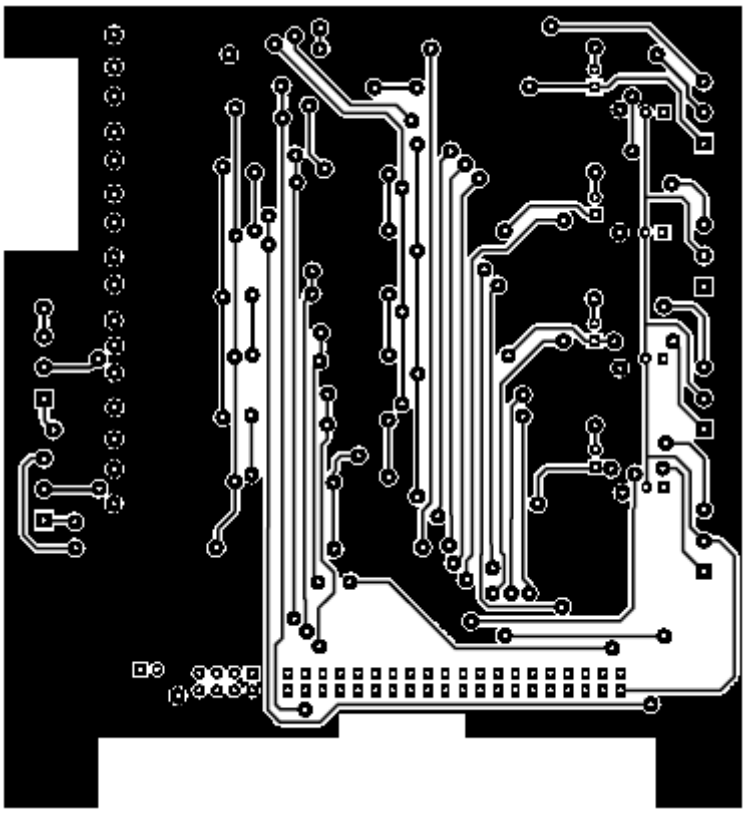


Figura 84: Face de "baixo" da PCB.

Apêndice B

B. Informação Adicional Sobre o Hardware da TMU

B.1.Placa de Interface

Esta placa foi projectada para permitir o interface da TMU com os barramentos CAN. Foram utilizados, uma vez mais, quatro *transceivers* SN65HVD232 da *Texas Instruments*.

É possível, por intermédio de conectores ligar ou desligar resistências de terminação de 120Ω, existindo uma resistência de terminação para cada *transceiver*.

Na figura seguinte está representado o circuito, onde é possível ver também a correspondência entre os pinos do *expansion header*, com os pinos da FPGA. Na Figura 86 e Figura 87 estão apresentadas as duas faces da PCB deste circuito de interface. Na

Tabela 23 está representada a relação que existe entre os pinos do *header* de expansão e os pinos da FPGA.

Expansion Header Pins	**FPGA *Pins***
4	E11
6	D11
8	E12
10	D12
12	D13
14	B14
16	C15
18	B15

Tabela 23: Correspondência entre os pinos do *header* de expansão e os pinos da FPGA.

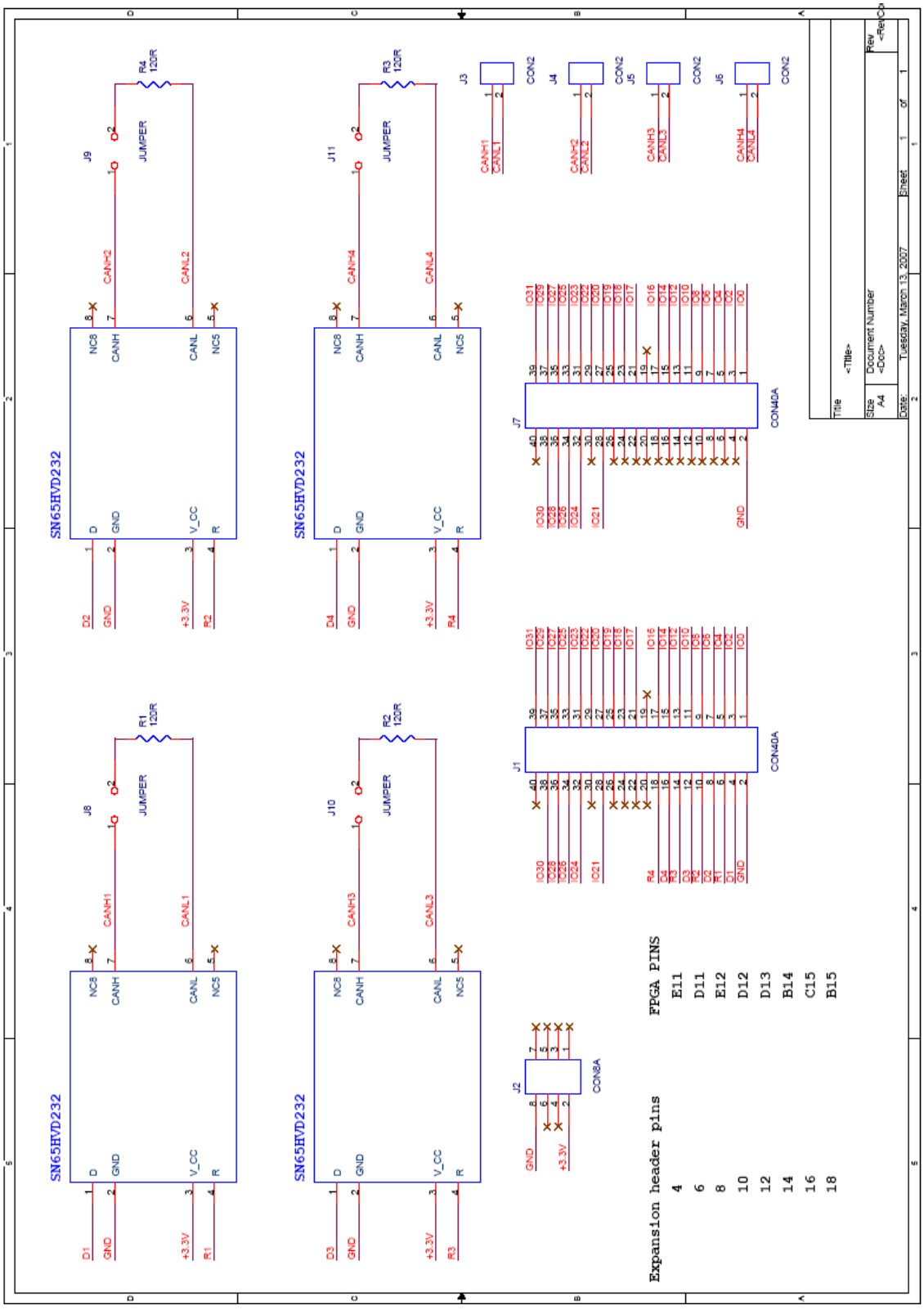


Figura 85: Circuito da placa de interface da TMU.

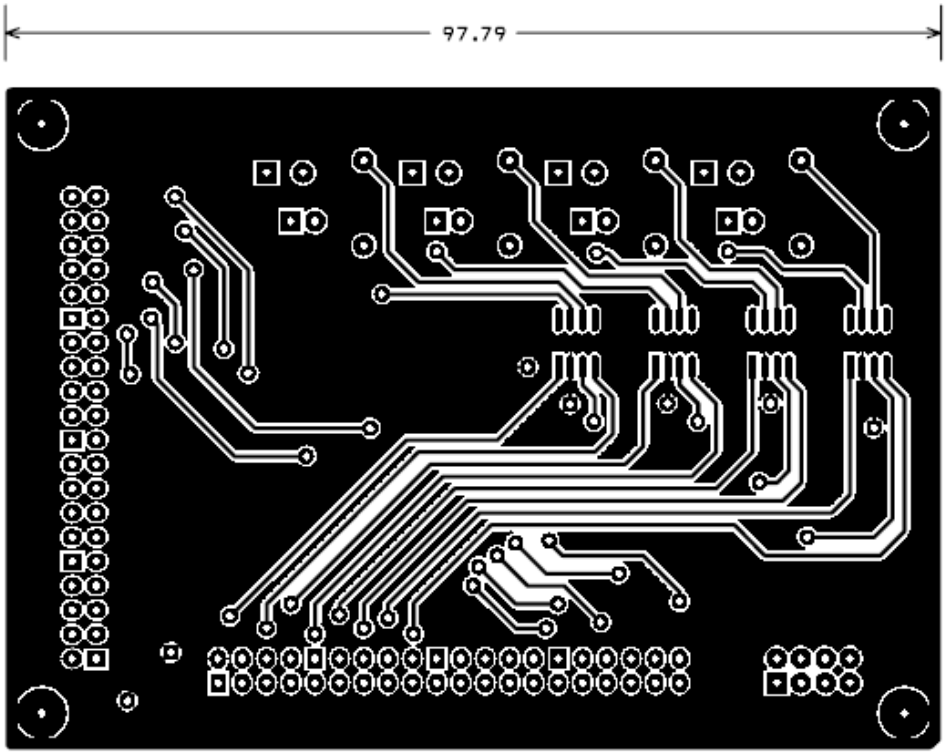


Figura 86: Face de “cima” da PCB (dimensões em milímetros).

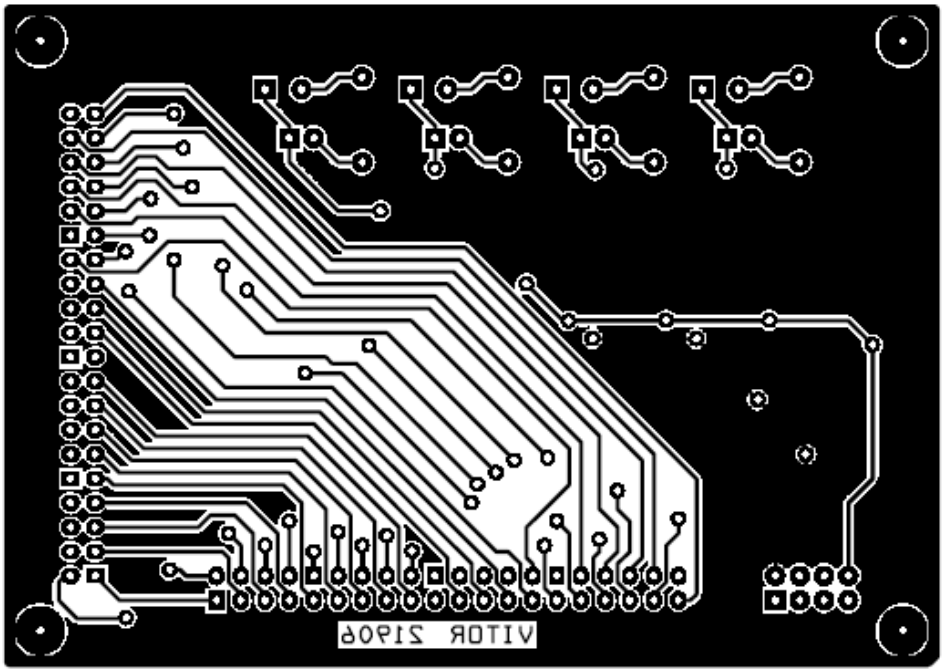


Figura 87: Face de "baixo" da PCB.

Apêndice C

C. Programa de Monitorização PCAN

O programa de monitorização *PCAN-View*, é um programa que permite monitorizar o fluxo de dados numa rede CAN, Figura 88. Permite a conexão de aplicações do Windows a um barramento CAN ligado em tempo real a um PC.

O *PCAN-View* possui as seguintes características:

- Exibe o *ID* da mensagem recebida, tamanho e dados numa lista. Além disso, exibe o número de mensagens com a mesma identificação, e o atraso entre duas destas últimas mensagens. Portanto, a frequência das mensagens recebidas periodicamente pode ser obtida.
- Exibe as tramas remoto recebidas, assim como o número de vezes que foi recebida e o intervalo de tempo da sua recepção.
- Mensagens criadas arbitrariamente podem ser colocadas numa lista de transmissão. Estas mensagens podem ser enviadas automaticamente, quer em intervalos fixos, como uma resposta a uma trama remoto, ou manualmente.
- Erros ocorridos no barramento CAN ou no controlador são mostrados.

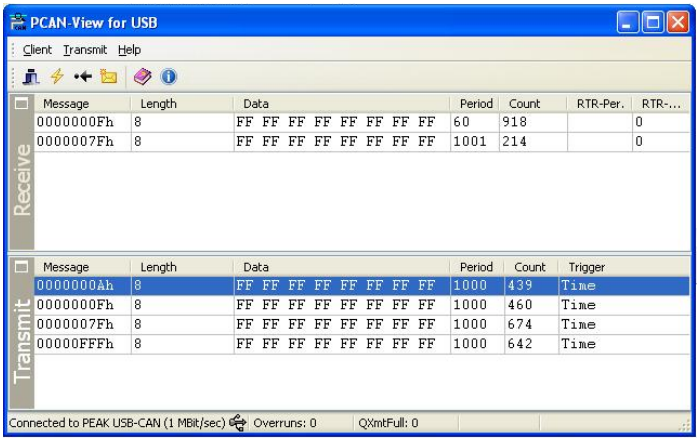


Figura 88: *PCAN-View*, programa de monitorização (*sniffer* CAN).

O *PCAN-View* permite inserir várias mensagens de transmissão criadas arbitrariamente. Como se pode ver na Figura 89, é possível editar o ID, o tamanho do campo de dados e o período. É possível indicar se as tramas são *extended* ou do tipo remoto. Se o período for de 0ms, as mensagens são enviadas manualmente, sendo necessário pressionar a tecla *space* do PC para fazer o respectivo envio.

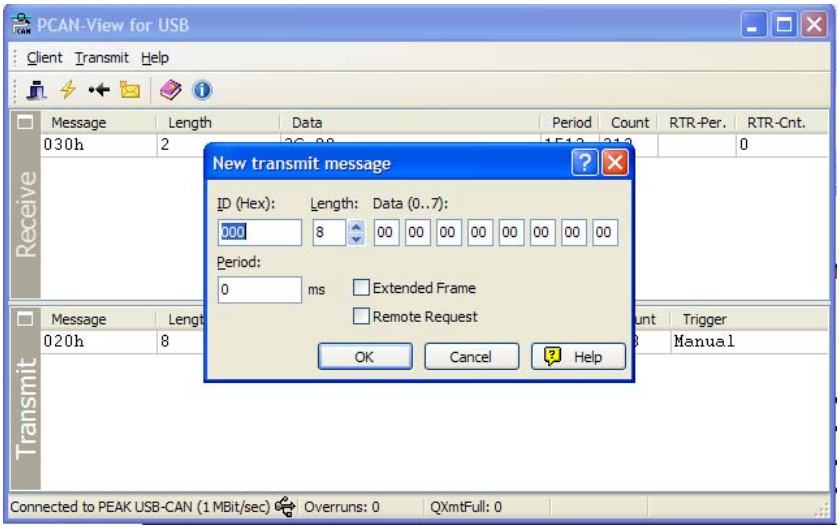


Figura 89: Introduzir nova mensagem de transmissão no *PCAN-View*.

Na Figura 90 encontra-se um exemplo de como inserir uma nova mensagem, neste caso a **SET_ADDRESS**.

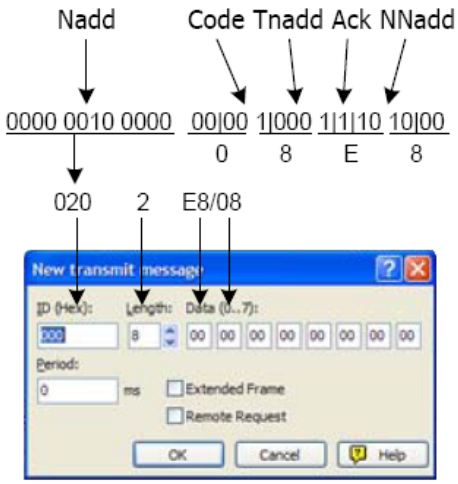


Figura 90: Exemplo de envio da mensagem *SET_ADDRESS*.

D. Lista de Acrónimos

ABS – *Anti-locking Brake Systems*

ACIUB – *Acceptance Identifier/Upper Bound*

ACK – *Acknowledge*

ACKEC – *Acknowledge Error Count*

ACMLB – *Acceptance Mask/Lower Bound*

ALU – *Arithmetic logic unit*

BITEC – *BIT Error Count*

BUSTM – *Bus Timing*

CAN – *Controller Area Network*

CMOS – *Complementary Metal-Oxide-Semiconductor*

CNTRL – *Control*

COMDR – *Command*

CRC – *Cyclic Redundancy Check*

CRCEC – *CRC Error Count*

CSMA/CA – *Carrier Sense Multiple Access with Collision Avoidance*

CSMA-CD – *Carrier Sense Multiple Access with Collision Detection*

CSMA-DCR – *Carrier Sense Multiple Access with Deterministic Collision Resolution*

DETI – Departamento de Electrónica, Telecomunicações e Informática

DIN – *Data IN*

DLC – *Data Lenght Code*

DOUT – *Data Out*

DRM – *Dynamic Redundancy Management*

DTM – *Dynamic Topology Management*

ECU – *Electronic control unit*

EOF – *End-of-Frame*

ESC - *Electronic Stability Control*

ESP – *Electronic Stability Program*

FIFO – *First In First Out*

FPGA – *Field Programmable Gate Array*

FRMEC – *Form Error Count*

FTU – *Fault-Tolerant Unit*

ID – *Identification*

IEEE – *Institute of Electrical and Electronic Engineers*

IENAB – *Interrupt Enable*

IIDENT – *Interrupt Identification*

ISO – *International Organization for Standardization*

MAC – *Media Access Control*

MIPS – Milhões de Instruções Por Segundo

NC – *Normaly Closed*

nMOS – *Negative Metal-Oxide-Semiconductor*

NO – *Normaly Open*

NRZ – *Non-Return-to-Zero*

NSU – *Network Switch Unit*

OSI – *Open Systems Interconnect*

PC – *Personal Computer*

PCB – *Printed Circuit Board*

PLC – *Power Line Communication*

pMOS – *Positive Metal-Oxide-Semiconductor*

RedCAN – *Redundant CAN*

RMSGC – *Rx Message Count*

RT – *Reconfiguration Table*

RXD03 – *Rx Message Data 03*

RXD47 – *Rx Message Data 47*

RXERC – *RX Error Count*

RXMCT – *Rx Message Control*

RXMID – *Rx Message Identifier*

RXTXS – *Rx/Tx Status*

SAE – *Society of Automotive Enginners*

SC – *Switch Controller*

SLIO – *Serial Linked I/O*

SOF – *Start-of-Frame*

STAT 0 – *Status 0*

STAT 1 – *Status 1*

STFEC – *Stuff Error Count*

TMR – *Triple Modular Redundancy*

TMSGC – *Tx Message Count*

TMU – *Topology Manager Unit*

TR – *Topology Reconfigurator*

TTCAN – *Time-Triggered* CAN

TTP/C – *Time-Triggered* Communication Protocol

TXD03 – *Tx Message Data 03*

TXD47 – *Tx Message Data 47*

TXERC – *TX Error Count*

TXMCT – *Tx Message Control*

TXMID – *Tx Message Identifier*

VHDL – *VHSIC Hardware Description Language*

VHSIC – *Very High Speed Integrated Circuits*

Bibliografia

Almeida, Luís. 2001. Sistemas Tolerantes a Falhas - Barramentos de Campo. EST -IPCB, 2º semestre 2001.

America, Department of Defense of United States of. 1978. Aircraft Internal Time Division Command/Response Multiplex Data Bus. Washington D.C., United States of America : s.n., 21 de September de 1978.

Arnaldo S. R. Oliveira, Nelson L. Arqueiro, Pedro N. Fonseca. CLAN – A Technology Independent Synthesizable CAN Controller. University of Aveiro / IEETA, Portugal : s.n.

Ashenden, Peter J. 1995. *The Designer's Guide to VHDL.* San Francisco : Morgan Kaufmann Publishers, 1995. ISBN 1-55860-270-4 .

—. 1990. *The VHDL CooKBook.* 1990.

Barranco, Manuel, Almeida, Luís e Proenza, Julián. 2005. ReCANcentrate: A replicated star topology for CAN networks. 2005. pp. pp. 469-476.

Barranco, Manuel, et al. 2004. CANcentrate: An active star topology for CAN networks. 2004. pp. pp. 219-228.

Celoxica. RC10 Manual.

CiA. 2001-2008. CAN in Automation. [Online] 2001-2008. http://www.can-cia.org/.

Cika. 2007. *Cika Electrónica S.R.L.* [Online] 2007. http://www.cika.com/productos/oki/.

Consortium, FlexRay. 2000. www.flexray.com. *FlexRay the communications sistem for advanced automotive control applicatinos.* [Online] 2000. http://www.flexray.com.

Controller Area Network Implementation in Microwave Systems. **Piotr Furmanski, Ziemowit Stolarski. 2002.** Wroclaw : s.n., 2002. Microwaves, Radar and Wireless Communications, 2002. MIKON-2002. 14th International Conference on. pp. 869- 873 vol.3.

Etshberger, Konrad. 2001. Controller Area Network. Weinheim , Alemanha : IXXAT Automation Press, 2001.

FlexRay. 2006. *Electrical Physical Layer Specification Version 2.1 Revision B.* 2006.

—. 2009. FlexRay the communication system for advanced automotive control applications. [Online] 2009. http://www.flexray.com/.

Instruments, National. 2006. Reed Relay Protection. *National Instruments.* [Online] 6 de Setembro de 2006. http://zone.ni.com/devzone/cda/tut/p/id/3953.

Instruments, Texas. 2006. SN65HVD232 datasheet. 2006.

IPUO, International P-NET User Organization. 1983. The P-NET Fieldbus . [Online] 1983. http://www.p-net.dk/.

ISO. 2003. *International Organization for Standardization.* [Online] 19 de 11 de 2003. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=33422.

—. 1947. International Organization for Standardization. [Online] 1947. www.iso.org.

—. 1993. ISO 11898-1:2003. *International Organization for Standardization.* [Online] 1993. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=33422.

—. 2003. ISO 15745-4:2003. [Online] 2003. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=33443.

ISO, 11898-1:2003. International Organization for Standardization. *International Organization for Standardization.* [Online] http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=33422.

Joaquim Ferreira, Luís Almeida, Member, IEEE, José Alberto Fonseca, Member, IEEE, Paulo Pedreiras, Member, IEEE, Ernesto Martins, Guillermo Rodríguez-Navas, Joan Rigo, Julián Proenza. 2006. "Combining Operational Flexibility and Dependability in FTT-CAN". *sinbad.ua.pt Publicações.* [Online] 5 de 2006. [Citação: 18 de 12 de 2008.] http://biblioteca.sinbad.ua.pt/publicacoes/19-2-2007_19-46-4190.

Kopetz, Hermann. 1997. *Real-Time Systems · Design Principles for Distributed Embedded Applications.* Boston : Kluwer Academic Publishers, 1997. ISBN 978-0-7923-9894-3.

Krishna, C. M. 1997 . Real-time systems / C. M. Krishna, Kang G. Shin . New York : McGraw-Hill , 1997 .

Larry L. Peterson, Bruce S. Davie. *Computer Networks: A Systems Approach.* s.l. : Morgan Kaufmann Publishers. ISBN: 1-55860-833-8.

Manuel Barranco, Julián Proenza, Guillermo Rodríguez-Navas, Luís Almeida. 2004. An Active Star Topology for Improving Fault Confinement in CAN Networks. 16 de Setembro de 2004.

MAXIM. 2008. What is a Transmission Gate (Analog Switch)? *MAXIM.* [Online] 2008. http://www.maxim-ic.com/appnotes.cfm/appnote_number/4243/.

MAXIM, Integrated Products. 2001. MAX4624/MAX4625 - 1Ohm, Low-Voltage, Single-Supply, SPDT Analog Switches. 2001.

MEDER. 2008. Reed Relays : CRF Relays. *MEDER Electronic.* [Online] 2008. http://www.meder.com.

Nissanke, Nimal. cop. 1997 . Realtime systems / Nimal Nissanke . s.l., Londres : Prentice Hall , cop. 1997 .

ODVA. 2008. ODVA. [Online] 2008. http://www.odva.org.

Profinet, Profibus &. 1989. The Industrial Communications Community Delivering Greater Enterprise Advantage. [Online] 1989. [Citação: 17 de 12 de 2008.] http://www.profibus.com/.

R. Pimentel, Juan e A. Fonseca, Jose. 2004. FlexCAN: A Flexible Architecture for Highly Dependable Embedded Applications. Kettering University Michigan USA, University of Aveiro Aveiro Portugal, 3rd Int. Workshop on Real-Time Networks to be held in Catania, Italy : s.n., July de 2004.

Rufino, José. 1997. *Redundant CAN Architectures for Dependable Communications.* Lisboa : s.n., 1997. Technical Report: CSTC RT-97-07.

Shaheen, Shehryar, Heffernan, Donal e Leen, Gabriel. 2003. A comparison of emerging time-triggered protocols for automotive X-by-wire control networks. *Proceedings of the I MECH E Part D Journal of Automobile Engineering.* s.l. : Professional Engineering Publishing, 2003, Vols. Volume 217, Number 1, pp. pp. 13-22(10).

Silva, Valter Filipe, Ferreira, Joaquim Castro e Fonseca, José Alberto. 2006. Dynamic Topology Management in CAN. 2006.

Sivencrona, Hakan, et al. 2004. RedCAN: Simulations of two Fault Recovery Algorithms for CAN. 2004.

Steve Corrigan, Texas Instruments. 2008. Controller Area Network Physical Layer Requirements. January de 2008.

Touloupis, Emmanuel, A Flint, James e D Ward, David. 2003. Safety-Critical Architectures for Automotive Applications. Loughborough University.

TTA-Group. 1998. TTA-Group The Cross-Industry Consortium for Time-Triggered Systems. [Online] 1998. http://www.ttagroup.org/technology/ttp.htm.

Veríıssimo, Paulo, Arroz, Guilherme e Rufino, José. 1997. A Columbus' Egg Idea for CAN Media Redundancy. Lisboa : s.n., 1997.

WorldFip. 1993. WorldFip Fieldbus. [Online] 1993. http://www.worldfip.org/.

Xilinx. 2005. PicoBlaze 8-bit Embedded Microcontroller User Guide. 21 de November de 2005.

—. 2008. Xilinx. [Online] 2008. http://www.xilinx.com.

YAMAR. 2007. DCAN250 - CAN over Battery. *DCAN250 - CAN over Battery.* 2007.