UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

FABIANO COSTA CARVALHO

Uma Extensão do Protocolo CAN para Aplicações Críticas em Sistemas Distribuídos

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Carlos Eduardo Pereira
Orientador

Porto Alegre, maio de 2006

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Carvalho, Fabiano Costa

Uma Extensão do Protocolo CAN para Aplicações Críticas em Sistemas Distribuídos / Fabiano Costa Carvalho. – Porto Alegre: PPGC da UFRGS, 2006.

113 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2006. Orientador: Carlos Eduardo Pereira.

1. Protocolos de tempo-real. 2. Sistemas embarcados. 3. Tolerância à falhas. 4. Sistemas distribuídos. I. Pereira, Carlos Eduardo. II. Título.

“It is highly desirable to ensure that the separation between theory and practice is minimized.”

— JONATHAN P. BOWEN

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ASIC	*Application Specific Integrated Circuit*
AT	Action Time
BIU	Bus Interface Unit
CAN	Controller Area Network
CASCA	Communication Architecture for Safety-Critical Applications
CC	Communication Frame
CMU	Communication Unit
CSMA	Carrier Sense Multiple Access
DACAPO	Dependable Architecture for Control Applications with Periodic Operation
DCA	Daisy-Chain Algorithm
EC	Embedded Controller
FC	Função de Convergência
FIFO	First-In First-Out
FPGA	Field Programmable Gate Array
GPS	Global Positioning System
IM	Intermodule Memory
ISO	International Organization for Standardization
IP	Intellectual Property/Internet Prototol
MMC	Mínimo Múltiplo Comum
MTTF	Mean Time To Failure
NTU	Network Time Unit
OSI	Open Systems Interconnection
PDA	Personal Digital Assistant
PLL	Phase Locked Loop
RM	Reference Message
RT	Real-Time

RTL	Register Transfer Level
RTOS	Real-Time Operating System
SAE	Society of Automotive Engineers
SC	System Cycle
SoC	System on a Chip
TDMA	Time-Division Multiple Access
TM	Table Memory
TT	Time-Triggered
TTA	Time-Triggered Architecture
TTCAN	Time Triggered Controller Area Network
TTP	Time-Triggered Protocol
TUR	Time Unit Ratio
UTC	Universal Time Coordinated

LISTA DE SÍMBOLOS

t	Tempo físico.
T	Tempo lógico.
ρ	Incerteza associada à um dispositivo oscilador.
δ_{max}	Precisão da base de tempo distribuída.
γ	Exatidão de um sistema distribuído.
ε	Erro associado à uma leitura de relógio.
Λ	Fator de correção.
κ	Fator de compensação.
δ_{ij}	Desvio entre os relógios de *i* e *j*.
α	Fator de amortização.
ϕ	Fator de normalização.
φ	Número máximo de slots de tempo vazios consecutivos.
R_{int}	Intervalo de sincronização.
M_i	Mensagem com identificador i.
L_{M_i}	Latência de M_i.

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Sistemas computacionais de tempo-real são tipicamente construídos a partir de primitivas de sincronização que fornecem uma noção do tempo no objetivo de coordenar a execução múltiplos fluxos de instruções em um processador. Quando o processamento é centralizado, a base de tempo destas primitivas é extraída do oscilador local da plataforma, permitindo que as ações do sistema sejam devidamente ordenadas, respeitando restrições de tempo e causalidade. No entanto, em sistemas distribuídos o problema não pode ser resolvido desta forma em decorrência de imperfeições nos dispositivos físicos. Diferenças mínimas na freqüência de osciladores fazem com que as bases de tempo dos componentes divirjam cada vez mais ao longo do tempo, dificultando ou até mesmo impossibilitando um ordenamento consistente de eventos. Por esta razão, sincronização de relógios é um serviço de fundamental importância, sobretudo em aplicações críticas, onde os níveis de confiabilidade exigidos são mais elevados. O presente trabalho consiste na proposta e implementação de uma plataforma de comunicação otimizada para sistemas de controle distribuídos, caracterizados por uma alta regularidade no comportamento da comunicação. O objetivo é propor uma solução em baixo nível com suporte para o projeto de sistemas distribuídos no domínio de aplicações críticas. A plataforma proposta, à qual foi atribuído o nome CASCA, sigla para "Communication Architecture for Safety-Critical Applications", é de fato uma extensão time-triggered do protocolo CAN. Acima da camada de enlace do protocolo original foram projetados mecanismos sincronização de relógios e criação inicial da base de tempo, implementados na forma de uma combinação de hardware e software. Principais características da plataforma são jitter mínimo, uma base de tempo global essencialmente distribuída e particionamento temporal. Diferentes alternativas de projeto foram consideradas, observando com maior atenção a viabilidade de prototipação em dispositivos FPGA para fins de validação e aplicação imediata em plataformas reconfiguráveis. Como forma de validação da plataforma, um sistema elementar formado por três nodos foi sintetizado com sucesso em bancada obtendo-se como resultado uma base de tempo essencialmente distribuída com precisão menor do que um micro-segundo.

Palavras-chave: Protocolos de tempo-real, Sistemas embarcados, Tolerância à falhas, Sistemas distribuídos.

An Extension of the CAN Protocol for Safety-Critical Applications in Embedded Systems

RESUMO

Real-time computing usually relies on synchronization primitives that provides an accurate notion of elapsed time with the intent to coordinate the execution of multiple instruction flows. When processing load is strictly centralized, the time source of these primitives is taken from local oscillator devices in order to ensure temporal and causal ordering of system actions. Nevertheless, in distributed systems the solution for this problem is not straightforward mainly due to imperfections of physical devices like subtle frequency drifts. Since oscillator frequencies are never exact, the time base of distributed components will diverge more and more as time progresses. Hence, it becomes clear that clock synchronization is indeed a service of utmost importance, specially in safety-critical in which the use of active redundancy is required to attain high dependability degrees. This work consists in a proposal and implementation of an optimized communication platform for distributed systems with regular communication behavior demands, which is a typical characteristic of distributed control systems. One of the main goals is to provide a solution that could offer adequate support for the design of safety-critical distributed systems which are know to be present in many sectors of modern society, including embedded automotive systems and industrial environments. The proposed platform is named CASCA, an acronym for "Communication Architecture for Safety-Critical Applications" and it is a time-triggered extension of the original CAN protocol over which the Daisy-Chain clock synchronization algorithm and an original procedure for initial time base creation were implemented as a combination of software and hardware blocks. Primal characteristics of the platform are essentially distributed global time, minimal network jitter and time composability. Different design alternatives were considered with special attention to the viability of fast prototyping using FPGA devices for the sake of validation and immediate use in reconfigurable platforms. In order to validate the proposed platform, an elementar network consisting of three nodes was successfully prototyped resulting in a clock-synchronized distributed system with a time base precision of less than one micro-second.

Palavras-chave: Real-time protocols, Embedded systems, Fault-tolerance, Distributed systems.

1 INTRODUÇÃO

1.1 Delimitação do Problema

Já há algum tempo, sistemas distribuídos têm sido construídos e utilizados em larga escala, em uma vasta diversidade de aplicações. Apesar de muitos destes sistemas funcionarem de maneira transparente, podendo até mesmo nem serem notados pela grande maioria daqueles que se favorecem de seus serviços, eles têm fundamental importância do ponto de vista econômico e social pois estão presentes em muitos setores da sociedade moderna. Sua utilização na indústria está diretamente ligada ao aumento da capacidade produtiva ao passo que em outras áreas eles têm contribuído de forma significativa para uma melhoria da qualidade de vida da população em geral. Nos dias de hoje, sistemas distribuídos são essenciais em ambientes industriais para o controle e monitoramento de linhas de montagem, usinas de geração de energia elétrica, refinarias de petróleo, ou qualquer outro processo elétrico, químico, hidráulico ou mecânico onde existe a necessidade de processamento de informação no formato digital. Além disso, sistemas distribuídos também estão cada vez mais presentes em sistemas embarcados de grande porte, como é o caso de aeronaves, automóveis e veículos de transporte em geral. Atualmente 20 a 30% do custo dos veículos está associado ao projeto e fabricação de sistemas eletrônicos muitos dos quais exercem funções de controle e diagnóstico integrado (POLEDNA; NIEDERSUSS; KROISS, 2000). Destaca-se neste domínio o uso de tecnologia por-fio (*by-wire*), ou seja, processadores e acionadores elétricos interligados em rede que cooperam na realização de funções de controle específicas, como por exemplo controle dinâmico de direção, suspensão, câmbio, freio e motor.

Existe de fato um forte interesse na indústria automotiva em fabricar veículos populares com sistemas distribuídos embarcados. Com a substituição de dispositivos de acionamento hidráulico e mecânicos convencionais por componentes eletrônicos é possível chegar a uma diminuição considerável de peso dos veículos (JANG; PARK; HAN, 2003). Veículos mais leves consomem menos energia (química) logo, têm maior autonomia. Além disso, algoritmos podem ser desenvolvidos para aumentar a segurança dos usuários auxiliando-os em situações de perigo (DILGER et al., 2005). Um sistema de controle de freios, por exemplo, poderia executar um algoritmo de redistribuição de carga caso o circuito de uma das rodas porventura venha a falhar (KOPETZ; ADEMAJ; HANZLIK, 2004). No caso de aeronaves, o maior desafio é construir sistemas altamente integrados buscando também a diminuição de peso, economia de recursos, além do compartilhamento de uma maior quantidade de informações, conforme destacado em (RUSHBY, 2001) e (ROSTAMZADEH et al., 1995).

Uma característica comum de todas estas aplicações é a distribuição espacial dos pontos de interesse no objeto controlado. Em automóveis e demais veículos de médio e

grande porte, os pontos de amostragem e atuação estão fisicamente separados por distâncias na ordem de metros. No caso de plantas industriais, estas distâncias podem atingir dezenas, senão centenas de metros em casos extremos. Sabe-se no entanto que todos estes pontos de processamento necessitam se comunicar de alguma forma na realização de uma determinada função em conjunto. Considerando a alta quantidade e diversidade dos dispositivos empregados, o uso de um barramento serial digital representa a solução mais eficiente devido ao baixo custo associado além da facilidade de projeto e manutenção. Contudo, o uso de um barramento compartilhado implica na adoção de um protocolo de acesso determinístico uma vez que a maioria das aplicações embarcadas e industriais estão diretamente envolvidas com a dinâmica do objeto ou processo controlado. O sistema deve ser capaz de reagir em tempo adequado a eventos e variações de grandezas físicas de interesse mesmo em situações de falha ou de pico de carga, caso contrário existe o risco de perder o controle exercido sobre a aplicação.

Tipicamente, sistemas determinísticos – também chamados de sistemas de tempo-real – são construídos a partir de primitivas de sincronização que fornecem uma noção do tempo transcorrido desde sua inicialização. Quando a carga de processamento é centralizada, a base de tempo destas primitivas é extraída diretamente do oscilador local da plataforma de hardware. Em sistemas monotarefa, a seqüência de execução das ações fica explicitamente definida no código uma vez que existe somente um fluxo de instruções a ser executado pelo processador. Aplicações mais complexas são mapeadas para uma plataforma multitarefa como por exemplo um RTOS na finalidade de particionar e ordenar o tempo de execução das tarefas e fazer com que as ações do sistema respeitem os requisitos de tempo e causalidade, que são geralmente expressos na forma de instantes de ativação e prazos máximos para término e entrega de resultados.

Em se tratando de sistemas distribuídos a consistência temporal se torna mais complicada, principalmente pelo fato das fontes primárias de tempo de cada nodo serem independentes. Pequenas flutuações na freqüência de dispositivos físicos causam um escorregamento entre relógios que aumenta cada vez mais no decorrer do tempo. Na prática, um oscilador de cristal pode apresentar um erro absoluto de até 10^{-5} a cada segundo, o que equivale a um erro de 0.86 segundos a cada dia conforme destacado em (SURI; HUGUE; WALTER, 1994). No pior caso, ao final de um dia, dois processos distribuídos podem estar observando bases de tempo com uma diferença de 1.72 segundos. Por isso, sistemas distribuídos devem ser dotados de mecanismos de sincronização mais elaborados para que as suas tarefas possam cooperar na execução de alguma função. No capítulo 3 esta questão será discutida de maneira mais detalhada, dando mais ênfase na sincronização baseada na correção de relógios lógicos, que tem como maior vantagem o fato de ser possível tratar restrições e relações de tempo de maneira global.

1.1.1 Sistemas Críticos

Conforme destacado em (AVIZIENIS, 1995), quanto maior o benefício oferecido por um sistema computacional, maiores são os danos causados em caso de falha do mesmo. Dentro do domínio de aplicações embarcadas ou industriais são muitas as situações onde uma falha pode ter conseqüências catastróficas, podendo até mesmo resultar na perda de vidas. Em aplicações industriais, uma falha pode representar perdas materiais, prejudicar o meio ambiente ou até mesmo colocar em risco a integridade física de operários. Já em sistemas embarcados existe um envolvimento direto com a segurança do condutor e dos demais usuários do veículo.

O projeto de um sistema de tempo-real se preocupa com a pontualidade das ações e o

correto fornecimento de resultados. Um dos principais desafios é garantir atrasos máximos mesmo em momentos de alta demanda. O mais importante de fato é avaliar previamente se os mecanismos de controle de acesso a recursos compartilhados não apresentam situações de travamento, caso contrário pode existir a possibilidade de alguma tarefa entrar em estado de inanição levando a uma paralisação parcial ou total do sistema. Cabe também destacar que um sistema pontual nem sempre é um sistema rápido. Sabe-se que alta velocidade na comunicação e no processamento das tarefas não garante que um sistema seja determinístico. Muitas aplicações possuem uma dinâmica bastante lenta, como é o caso de processos químicos onde os períodos de amostragem são na ordem de centenas de milisegundos. No benchmark da SAE (Sociedade dos Engenheiros Automotivos) os requisitos de comunicação mais críticos são de 8 bits a cada 5ms (TINDELL; BURNS, 1995). Em sistemas por-fio, as freqüências de controle foram estimadas na ordem de 100Hz (BRIDAL; SNEDSBØL; JOHANSSON, 1994). Assumindo que mensagens carregam variáveis de processo de 32 bits no máximo, então a vazão total demandada por cada laço de controle individual fica limitada a somente 3.2kpbs, o que é muito pouco em comparação às demandas, digamos, em sistemas de telecomunicação ou redes de escritório.

Por outro lado, quando o funcionamento de um sistema está diretamente envolvido com a segurança de seres humanos ou de propriedade de alto valor agregado, então este sistema é classificado como "crítico" (PALANQUE; PATERNO; WRIGHT, 1998) e seu projeto deve levar em conta não somente a pontualidade das ações do sistema como também deve se preocupar em garantir que seja pontual mesmo em caso de falhas. Sistemas críticos devem ser construídos visando garantir uma menor variação possível no seu comportamento em tempo de operação. Esforços devem ser desprendidos em todas as etapas do projeto para que o sistema siga fiel à sua especificação durante o maior período possível mesmo que com uma certa degradação na qualidade dos serviços oferecidos.

No contexto deste trabalho o termo "embarcado" se refere a um sistema computacional de propósitos específicos para uma determinada aplicação, que se desloca no espaço e que experimenta diferentes situações de funcionamento. Logo, além de toda a dinâmica e requisitos de tempo-real envolvidos, deve-se chamar a atenção para outras duas questões: Primeiro, tendo em vista que o sistema se desloca no espaço, sua fonte de alimentação possui uma capacidade limitada de armazenamento energético. Neste caso, busca-se o menor consumo possível para que o tempo de disponibilidade contínua de serviços seja maior. Segundo, é importante lembrar que algumas das diferentes situações de funcionamento podem colocar alguns componentes do sistema em risco. Aeronaves são constantemente submetidas à tempestades eletromagnéticas e mudanças bruscas de temperatura enquanto que automóveis estão sujeitos a colisões, corrosão, etc. O envelhecimento de circuitos integrados e demais componentes eletrônicos também é um problema característico deste domínio. É necessário prever estas condições adversas em tempo de projeto para garantir que em caso de falha o sistema siga operando, mesmo que com algum nível de degradação na qualidade de serviços.

Sistemas críticos embarcados são característicos de sistemas embarcados de grande porte, classificação em contraste com a de sistemas de pequeno porte, como por exemplo celulares, câmeras digitais, PDAs, entre outros, onde uma falha tem como conseqüência somente um leve descontentamento do usuário final. Em ambos os casos, a economia de energia é considerada meta básica de projeto porém, a dissipação média de potência em sistemas de grande porte é centenas, senão milhares de vezes maior quando comparado com valores equivalentes do segundo grupo. Em automóveis, o consumo de potência dos

circuitos integrados representa uma parcela mínima do consumo total. Observa-se que o consumo médio em sistemas embarcados de grande porte depende muito pouco da energia gasta para alimentar processadores, memória e controladores de rede. Um alternador convencional é capaz de fornecer correntes na ordem de 100 amperes que é em maior parte utilizada por acionadores eletromecânicos, sistema de iluminação e audio. Por estão razão, o trabalho realizado tem seu foco nos aspectos de determinismo e confiabilidade de sistemas distribuídos, ainda que algumas decisões de projeto também tenham sido influenciadas por fatores relacionados ao consumo energético.

Observa-se até então duas questões conflitantes. Por um lado, sistemas não devem falhar e por outro, ambientes e condições de operação hostis introduzem constantemente falhas no sistema. Outro fator complicador é que a probabilidade de ocorrência de falhas estruturais e de projeto cresce com o aumento da complexidade dos circuitos integrados, i.e. número de transistores, células lógicas e bits de memória por unidade de área. O fato é que para atingir níveis de segurança aceitáveis em sistemas críticos a aplicação de técnicas de tolerância à falhas é imprescindível. Em se tratando de sistemas distribuídos, todas estas preocupações se transferem diretamente para a plataforma de comunicação a qual deve oferecer suporte adequado para a implementação de mecanismos de redundância baseados na replicação de componentes funcionais e não-funcionais.

1.2 Motivação

Assumindo que as unidades de processamento em um ambiente distribuído se comunicam através de um mesmo barramento serial, o componente mais importante é sem dúvida a plataforma de comunicação, definida como sendo o conjunto formado pelo protocolo juntamente com todos os dispositivos físicos empregados tais como transdutores, cabos e conectores. Contudo, a grande maioria dos protocolos de acesso disponíveis hoje no mercado sofrem de uma falta de previsibilidade nos serviços oferecidos e oferecem suporte bastante limitado ou até mesmo inexistente para o uso de técnicas de tolerância à falhas. Apesar de ser largamente difundido, o mecanismo de controle de acesso utilizado pelos adaptadores Ethernet não é adequado para sistemas críticos. A estratégia usada para evitar colisões, conhecida como back-off exponencial binário, se baseia em um controle de acesso de caráter aleatório logo, existe sempre uma probabilidade não-nula que colisões ocorram indefinidamente impedindo a transmissão do quadro. Maiores detalhes sobre o comportamento temporal do protocolo Ethernet podem ser encontrados em (KWAK; SONG; MILLER, 2003). O protocolo CAN desenvolvido pela Bosch e largamente utilizado na indústria automotiva possui um mecanismo de resolução de colisões não-destrutivo extremamente eficiente onde as transmissões são ordenadas de acordo com prioridades atribuídas às mensagens. Contudo, o mecanismo de detecção e tratamento de erros impede que o protocolo CAN seja utilizado na sua forma original em sistemas críticos, como é o caso de aplicações de controle embarcadas. Considerando a ocorrência de erros, mensagens de menor prioridade podem sofrer atrasos com variações excessivas fazendo com que malhas de controle apresentem comportamento instável (BROSTER; BURNS, 2001).

O projeto de sistemas críticos carece de suporte para o uso de técnicas de redundância ativa, considerada a única maneira de se atingir os elevados níveis de disponibilidade e confiabilidade exigidos neste domínio (DILGER et al., 2005). De acordo com Rushby (RUSHBY, 2001), em algumas aplicações o sistema deve apresentar uma taxa de ocorrência de erros menor do que 10^{-9} por hora, assumindo um conjunto de falhas que melhor

represente aquilo o que acontece na prática. Experiência em projetos anteriores tem mostrado que soluções mais robustas são possíveis se, em baixo nível, a plataforma oferece suporte adequado (AVIZIENIS, 1997). Por isso, é dado preferência para metodologias de projeto do tipo bottom-up, onde tolerância à falhas e determinismo se tornam metas a serem priorizadas em todas as fases do desenvolvimento, iniciando já na concepção dos componentes de hardware da plataforma tais como processadores e controladores de rede.

É iminente o interesse por parte da indústria e da comunidade acadêmica no desenvolvimento de novos protocolos, sobretudo quando o foco são aplicações de controle para sistemas críticos (SRINIVASAN; LUNDQVIST, 2002). Um dos maiores desafios está na escolha do conjunto de serviços mais adequado e na síntese deste conjunto fazendo uso otimizado de recursos de hardware e software, garantindo fidelidade entre comportamento previsto e comportamento em regime de operação. A utilização de um processador de uso geral para execução em software de um algoritmo de sincronização de relógios pode ser a solução mais eficiente devido ao alto custo associado a uma implementação em hardware. Em contrapartida, os atrasos causados pela execução destas funções devem ser considerados pois impactam diretamente na qualidade dos serviços oferecidos. A definição de uma plataforma de comunicação genérica, que possa ser utilizada em todos os casos, é certamente um esforço em vão pois cada domínio de aplicação possui características e requisitos próprios. Alguns sistemas necessitam de maior velocidade de transmissão na medida que outros demandam mecanismos de detecção de erros mais apurados e suporte para tolerância à falhas por razões de segurança. No que diz respeito ao comportamento da comunicação, alguns sistemas devem ser mais reativos no sentido de prover respostas rápidas a eventos externos enquanto que outros devem apresentar maior regularidade na transmissão de dados. Antes de mais nada, é essencial entender o problema e fazer um levantamento cauteloso dos requisitos da aplicação para que então se possa determinar primeiro, o modelo de comunicação mais adequado e em seguida, a arquitetura da plataforma de comunicação que irá suportar o mapeamento das tarefas do sistema.

O desenvolvimento de plataformas de comunicação para sistemas embarcados distribuídos é hoje estado da arte. Existem diversas alternativas de projeto propostas na literatura científica, com diferentes características de performance, confiabilidade e custo de implementação. Cada uma destas alternativas consiste em uma determinada combinação de funções que podem ser implementadas na forma de módulos de hardware e software. Muitas soluções, tanto acadêmicas quanto comerciais, têm sido propostas no intuito de possibilitar a construção de sistemas cada vez mais confiáveis. Como exemplo de plataforma comercial podemos citar o Protocolo Time-Triggered (TTP/C, 2003), resultado do projeto MARS que iniciou na Universidade de Viena, na Áustria, sob o comando de Hermann Kopetz. Apesar de ter sido originalmente concebido dentro da academia, atualmente controladores que implementam a versão tolerante à falhas, o TTP/C, são comercializados pela empresa TTTech, em escala de produção. Maiores detalhes referentes ao TTP/C serão apresentados na seção 4.1. Demais propostas mais significativas são analisadas no capítulo 4.

1.3 Objetivos do Trabalho

Este trabalho consiste na proposta e implementação de uma plataforma de comunicação otimizada, à qual foi atribuído o nome de CASCA, referente às iniciais de: **Communication Architecture for Safety-Critical Applications**. CASCA é basicamente uma extensão do protocolo CAN sobre o qual foi implementado um mecanismo de controle

de acesso TDMA acoplado a um algoritmo de sincronização de relógios conhecido como Daisy-Chain, que têm como objetivo oferecer uma base de tempo global para coordenar as diferentes tarefas que são executadas em processadores distribuídos. A principal finalidade da plataforma proposta é servir como infraestrutura de base para o mapeamento de aplicações de controle em sistemas embarcados de grande porte, como é o caso de automóveis, caminhões, barcos, trens, máquinas agrícolas, entre outros. Contudo, sua utilização não deve ficar restrita a este domínio, podendo ser aproveitada em aplicações diversas como por exemplo sistemas de controle e automação industrial.

A arquitetura do controlador de rede, i.e. do bloco lógico da plataforma, deve ser altamente modularizado, permitindo que CASCA possa ser configurada para aplicações mais simples com um subconjunto reduzido de recursos conforme será visto no capítulo 5. Em se tratando de aplicações críticas, CASCA pode ser configurada para operar sobre barramentos seriais redundantes, resultando em um aumento significativo da utilização de recursos para síntese dos controladores.

Atualmente, o reaproveitamento de projetos é considerado a melhor maneira de lidar com a grande diferença entre número de transistores que podem ser inseridos em um circuito integrado e a capacidade de projeto de ferramentas e engenheiros. Por isso, foi decidido usar a linguagem de descrição de hardware VHDL para o desenvolvimento de todo o bloco lógico visando sempre a portabilidade e a possibilidade de reuso no projeto de circuitos maiores e mais complexos tais como SoCs.

CASCA passa a fazer parte do repositório de componentes de propriedade intelectual do Laboratório de Sistemas Embarcados (LSE) do Instituto de Informática da Universidade Federal do Rio Grande do Sul. O fato de ter sido implementada na forma de uma descrição RTL usando a linguagem VHDL faz com que a CASCA possa ser facilmente portada para dispositivos de lógica programável de diferentes fabricantes. O código RTL da CASCA é inteiramente sintetizável e está disponível para uso imediato em protótipos usando FPGAs de baixa densidade. Além disso, o modelo de comunicação, a arquitetura e os serviços da CASCA devem influenciar diretamente decisões em alto nível no que se refere ao projeto de sistemas distribuídos.

2 BASE TEÓRICA

Este capítulo apresenta os principais conceitos e definições que formam a base teórica para este presente trabalho. Pelo tamanho do conteúdo teórico relacionado, o conceito de sincronização de relógios foi separado em um capítulo a parte. No decorrer do texto, o uso de caracteres em negrito será adotado para destacar os termos, definições e idéias mais relevantes na medida em que aparecem pela primeira vez.

2.1 Modelo OSI de 7 camadas

Com o propósito de padronizar o projeto e utilização de protocolos de comunicação, a ISO "International Organization for Standardization" criou o Modelo de Referência OSI "Open Systems Interconnection" que é resultado de uma fusão de idéias e interesses de diferentes usuários, administradores e fabricantes mais expressivos do momento da sua aprovação. O modelo OSI não define um protocolo universal mas uma organização de diferentes serviços de rede em uma hierarquia de 7 camadas, representadas na forma de uma seqüência vertical. Por conveniência, serão descritas a seguir as camadas do modelo OSI de modo resumido, na ordem ascendente desta hierarquia, começando pela camada de mais baixo nível.

1. **Física**

   Camada de mais baixo nível, responsável basicamente pela transformação de um fluxo binário em codificação elétrica e vice-versa, sem nenhum conhecimento sobre o conteúdo e formato dos quadros que são transmitidos ou recebidos.

2. **Enlace**

   Realiza a serialização e a paralelização de dados que passam pela interface com a camada física. Determina o formato dos quadros que trafegam pela rede e executa diversas funções como detecção de erros, bit (ou byte) stuffing, e controle de arbitração, para evitar colisões de acesso ao meio físico. A camada de enlace juntamente com a camada física implementam os serviços básicos para comunicação entre dois componentes fisicamente separados.

3. **Rede**

   Responsável basicamente pela definição de rotas, isto é, caminhos para transmissão de pacotes entre subredes. A definição de rotas pode ser tanto dinâmica quanto estática. No primeiro caso, dado um endereço de origem e um de destino a rota permanece a mesma ao longo do tempo ao passo que no segundo caso a rota pode mudar por razões de congestionamento, perda de conexões físicas, entre outros.

4. **Transporte**

   Realiza funções de segmentação de estruturas de dados que não cabem em uma só mensagem, controle de perda e duplicação de quadros, controle de fluxo, reordenamento de quadros recebidos fora de ordem, confirmação de recebimento, entre outros. A finalidade da camada de transporte é garantir a consistência na transmissão fim-a-fim de estruturas de dados mais complexas, em decorrência dos problemas que surgem pelo alto particionamento de uma rede.

5. **Sessão**

   Permite que aplicações em computadores distintos estabeleçam sessões de comunicação, por onde dados trafegam dentro de um determinado contexto. Em cada sessão, as aplicações envolvidas definem como será feita a transmissão de dados e são inseridas marcações nos dados transmitidos. Se porventura a rede falhar, ambas as máquinas reiniciam a transmissão dos dados a partir da última marcação recebida pelo computador receptor para restaurar a última sessão.

6. **Apresentação**

   Também chamada camada de tradução. Tem a função de converter o formato do dado recebido pela camada superior em um formato comum a ser usado na transmissão, ou seja, um formato entendido pelas camadas inferiores e vice-versa. Pode ser usada, por exemplo, para compressão ou criptografia automática de dados.

7. **Aplicação**

   A camada de aplicação consiste na interface entre o modelo OSI e as tarefas que estão sendo executadas em um determinado nodo da rede.

O modelo OSI estabelece que cada camada deve oferecer serviços para as camadas superiores e utilizar serviços das camadas inferiores. Dependendo do nível em que se deseja trabalhar, assume-se que as camadas se comunicam diretamente entre si, podendo ser abstraído todo o processamento que acontece nas camadas inferiores. Outra característica importante é que o tamanho da estrutura de dados básica do protocolo aumenta na medida que desce na hierarquia. Na descida, cada camada faz o encapsulamento das informações de controle utilizadas pelos serviços naquele nível e repassa um pacote de maior tamanho para a camada logo abaixo.

Por simplicidade, o conjunto de protocolos que implementa uma ou mais camadas do modelo OSI será chamado de **pilha de rede**, ou somente pilha, no decorrer do texto. Também, a indicação de uma camada pelo seu número de seqüencia ou pelo seu respectivo nome serão utilizadas de forma indiscriminada.

Interessante notar que, apesar de ser sempre referenciado, o modelo OSI quase nunca é seguido a risca. Muitos protocolos existentes não são fieis ao modelo OSI na sua implementação, principalmente pelo fato que, na prática, somente um subconjunto de funcionalidades previstas para cada camada é efetivamente utilizado. Além disso, nem sempre é obrigatória a implementação de todas as 7 camadas para se construir um sistema em rede. Em se tratando de redes industriais ou embarcadas, a maioria dos protocolos cobre somente as camadas 1, 2 e 7. As principais funcionalidades das demais camadas são implementadas ou pela aplicação, ou em nível 7, ou não são necessárias dadas as características arquiteturais do sistema.

Todavia, o estudo do modelo OSI é extremamente didático, e facilita o entendimento e comparação do funcionamento de protocolos desenvolvidos por diferentes fabricantes. Sua importância tem sido, portanto, muito maior enquanto ferramenta de estudo e análise do que como padrão de projeto, cumprindo seu propósito inicial, ou seja, servir como modelo de referência.

2.2 Protocolos de Tempo-Real

A pilha de rede do modelo OSI é formada por protocolos implementados em módulos de software e/ou dispositivos de hardware que executam serviços de comunicação para a camada da aplicação. No caso de aplicações de controle embarcadas ou industriais, estes protocolos devem possuir propriedades que os qualifiquem como protocolos de tempo-real, uma vez que **os requisitos de tempo da aplicação se transferem diretamente para a plataforma de comunicação utilizada**.

O tempo de resposta de um protocolo é definido como sendo o intervalo de tempo entre a solicitação de um serviço de rede e o instante efetivo de término do mesmo. Um protocolo ou qualquer outro tipo de sistema é dito de tempo-real, determinístico, ou previsível, se os tempos de resposta associados à execução de seus serviços podem ser previamente estipulados sobre um ponto de vista funcional, desconsiderando a princípio a ocorrência de falhas. Em se tratando de protocolos de comunicação, a análise temporal busca estabelecer, principalmente, atrasos máximos para entrega de mensagens através de uma análise de tudo o que acontece no caminho que a mensagem percorre desde sua criação até o seu destino.

Uma fonte de indeterminismo no caminho percorrido pelas mensagens é definida como qualquer condição de parada na pilha de rede que porventura venha a adiar o término da execução de algum serviço de maneira indefinida. Tomando como referência o modelo OSI, estas fontes existem em todas as camadas, o que torna a análise de características de tempo-real dos protocolos de camadas mais superiores uma tarefa um tanto complicada, onde uma série de pressupostos devem ser feitos, como por exemplo a restrição do número de nodos e segmentos de rede, número de pontes e roteadores intermediários, entre outros. Em (FELSER, 2001) o autor apresenta as dificuldades e problemas associados ao uso de Ethernet e TCP/IP como protocolo chão-de-fábrica em ambientes industriais.

Considerando que na maioria dos casos os protocolos embarcados e industriais implementam somente as camadas 1, 2 e 7, interessa-nos saber quais são as fontes de indeterminismo intrínsecas destas camadas.

Controle de fluxo

> O serviço de controle de fluxo em um protocolo serve para evitar que informações sejam perdidas pelo estouro da capacidade de armazenamento dos buffers de entrada ou de saída existentes nas camadas 2 e/ou 7. A tarefa que está produzindo os dados é notificada de que uma ou mais tarefas consumidoras estão com seus buffers cheios. A notificação é feita por uma linha dedicada ou pelo envio de quadros de controle. Esta tarefa deve então retardar a produção de novas mensagens até que as outras consigam liberar espaço nos seus buffers de entrada. Da mesma forma, buffers de saída podem encher quando muitas mensagens estão sendo transmitidas ao mesmo tempo na rede. O controle de fluxo é considerado uma fonte de indeterminismo pois o atraso gerado é influenciado pela velocidade de outros nodos, e também pela carga de mensagens na rede em um dado momento, ambos fatores que

dificultam a análise do tempo de resposta do protocolo.

Filas

O uso de filas do tipo FIFO (*first-in, first-out*) não é adequado para protocolos de tempo-real. Isto porque em determinadas situações uma mensagem menos prioritária pode bloquear a saída da fila e impedir que mensagens mais críticas – que entraram depois – sejam transmitidas. Situações como esta acontecem em protocolos cujo acesso ao meio é baseado em prioridade de mensagens, tal é o caso dos protocolos CAN (CIA, 1999) e *byteflight* (BERWANGER; PELLER; GRIESSBACH, 2000). Para contornar este problema algum mecanismo adicional de controle, por exemplo um controle de prioridades na própria fila, deve ser implementado. Uma mensagem de maior prioridade deve poder tomar a frente de uma menos prioritária e portanto ser transmitida antes, sem a necessidade do esvaziamento da fila.

Controle de acesso

Assumindo que os nodos de um sistema distribuído estão todos conectados sobre um mesmo barramento serial, o controle de acesso ao meio têm a função de arbitrar o tempo de acesso para evitar colisões. A política de arbitração de um protocolo de tempo real deve garantir que todas os quadros sejam efetivamente entregues ao respectivo destino dentro de certos limites de tempo. Em muitos casos, a política de arbitração utilizada induz atrasos imprevisíveis que dificultam ou até mesmo impossibilitam a avaliação destes limites nos prazos de entrega, conforme será discutido a seguir.

2.2.1 Suporte ao Particionamento Temporal

Devido à grande quantidade e heterogeneidade dos dispositivos eletrônicos empregados atualmente em sistemas distribuídos, o uso de um barramento serial como topologia de rede é considerada a solução mais adequada, principalmente por razões de custo e facilidade de projeto. Além disso, o uso de um barramento serial permite um maior compartilhamento e integração de informações, uma vez que toda mensagem transmitida pode ser capturada por qualquer outro dispositivo conectado à ele. Contudo, pode acontecer por exemplo de em caso de falha um nodo tentar monopolizar o acesso ao meio físico transmitindo mensagens espúrias sem nenhum significado prático. Em outra situação de falha, um nodo qualquer pode tentar confundir as tarefas que estão sendo executadas em outros nodos transmitindo mensagens falsas, mas que no entanto são interpretadas no lado dos receptores como corretas.

Supondo agora que o sistema esteja livre de falhas, cada nodo transmite novas mensagens produzidas vão sendo transmitidas de acordo com uma seqüencia definida pela política de arbitração implementada na camada de enlace do protocolo. Seja ϕ a carga da rede em um dado instante, dada em mensagens por segundo, e L_{M_i} a latência máxima de transmissão esperada que uma mensagem M_i pode sofrer, dada em segundos. Para que o comportamento temporal de uma função distribuída (por exemplo um laço de controle do tipo sensor-atuador) não seja interferido pela integração de novos nodos ao barramento, deve-se garantir que L_{Mi} seja independente de ϕ, ou seja, **alterações no comportamento da carga da rede não devem influir na latência de transmissão das mensagens**.

Apesar de estar diretamente ligado à questão do determinismo, o conceito de particionamento temporal é mais amplo. Estritamente falando, refere-se à possibilidade de desenvolver subsistemas de maneira independente, para depois integrá-los sobre de um

mesmo canal de comunicação. Após a integração, tem-se que o funcionamento de um não irá interferir no de outro no que se refere à pontualidade das tarefas distribuídas. O esquema mostrado na Figura 2.1 propõe que a solução final a ser integrada em um sistema embarcado de grande porte (por exemplo um veículo) pode ser uma composição de diversos subsistemas, desenvolvidos e testados por fabricantes diferentes, no caso A, B e C. Se o protocolo de comunicação utilizado não oferece particionamento temporal, o subsistema do fabricante B pode interferir no funcionamento do subsistema de C e vice-versa. Seria então necessário analisar e testar todas as possíveis combinações de estados do sistema completo antes de colocá-lo em operação, para garantir que o comportamento temporal dos subsistemas não foi corrompido.

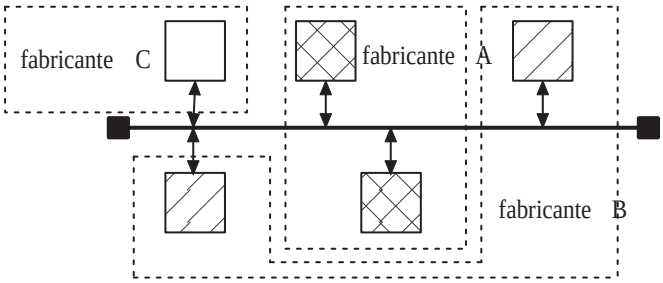

Figura 2.1: Composição de subsistemas.

Estratégias de controle de acesso do tipo mestre-escravo e passagem de token não garantem particionamento no tempo, apesar de serem largamente utilizadas em diversos protocolos industriais como por exemplo os protocolos Profibus (TOVAR; VASQUES, 1999) (PROFIBUS, 1992) e Modbus (MODBUS, 2004). Em sistemas mestre-escravo, onde o controle de acesso é estritamente centralizado, uma falha no mestre resulta na paralisação total da comunicação. Além disso, o tempo de resposta de uma função distribuída, por exemplo um laço de controle, está diretamente ligado ao tempo de varredura da rede que depende por sua vez do número de nodos escravos presentes no barramento.

No caso de estratégias do tipo passagem de token, existe o perigo de perda do token ou por motivo de retenção em algum dos nodos ou por falha no barramento, o que irá desencadear um processo de formação do anel lógico e criação de novo token que envolve todos os nodos, durante o qual a troca de mensagens da aplicação fica suspensa. Além disso, o tempo de rotação do token também é função direta do número de nodos.

Por outro lado, estratégias de multi-acesso representam aparentemente uma solução mais adequada pois a princípio qualquer nodo tem a permissão de tentar utilizar o barramento quando necessário, sem necessitar de nenhum tipo de autorização por parte de outro nodo. Neste caso a comunicação do sistema nunca será suspensa pela destruição ou falha de um ou mais componentes. Todavia, existe sempre o risco de que colisões ocorram fazendo com que a transmissão das mensagens que colidiram seja postergada. Isto é exatamente o que acontece com o mecanismo CSMA/CD do protocolo Ethernet no qual o comportamento aleatório do algoritmo de retransmissão executado em caso de colisão impede qualquer avaliação determinística de tempos de resposta. No protocolo Ethernet, os quadros têm uma probabilidade de serem transmitidos que é inversamente proporcional ao número de tentativas. Como conseqüência, surgem efeitos conhecidos pelo nome de **efeito captura**, quando um nó monopolisa o uso do barramento, e **inanição de quadro**, quando um quadro é descartado após 16 tentativas de transmissão sucessivas. O uso de switches pode eliminar a ocorrência de colisões porém, novos problemas

surgem, o mais notável sendo o aumento de custo de implementação. Ao invés de um par de fios que conecta N nodos, o barramento neste caso passa a consistir em N cabos full-duplex conectados a um componente adicional com capacidade de processamento e memória própria. Além disso, conforme destaca Song (SONG, 2001), a análise do tempo de resposta fica bastante dificultada pela falta de padronização de arquiteturas de switches. Segundo o autor do artigo, para cada arquitetura existem três fatores que impactam diretamente no determinismo da pilha de rede: a capacidade de processamento do switch, a política de redirecionamento de quadros e o congestionamento de portas de saída.

Uma parcela considerável da pesquisa sobre protocolos para sistemas embarcados altamente confiáveis está focada atualmente no modelo de comunicação time-triggered, ou somente TT, o qual será apresentado em maiores detalhes no capítulo 4. Existe um consenso na comunidade científica indicando que este é um paradigma altamente eficiente para projeto de sistemas críticos conforme é possível observar em (LONN; SNEDSBØL, 1995), (RUSHBY, 2001) e (ALBERT, 2004).

No modelo TT, o controle de acesso é feito por multiplexagem do tempo, também chamado de método TDMA (Time-Division Multiple Access). Este controle de acesso por sua vez está fortemente acoplado a um mecanismo de sincronização de relógios que fornece uma base de tempo global, visível por todos os nodos dos sistema, a partir da qual são definidas janelas de tempo exclusivas – **slots** – dentro das quais cada nodo deve transmitir uma ou mais mensagens sem o risco de contenção. Assim, sob uma perspectiva funcional, o particionamento no tempo é garantido pois colisões não existem além do fato de que a queda de um ou mais nodos não influi no funcionamento dos restantes, na condição que o algoritmo de sincronização de relógios utilizado seja capaz de tolerar este tipo de situação.

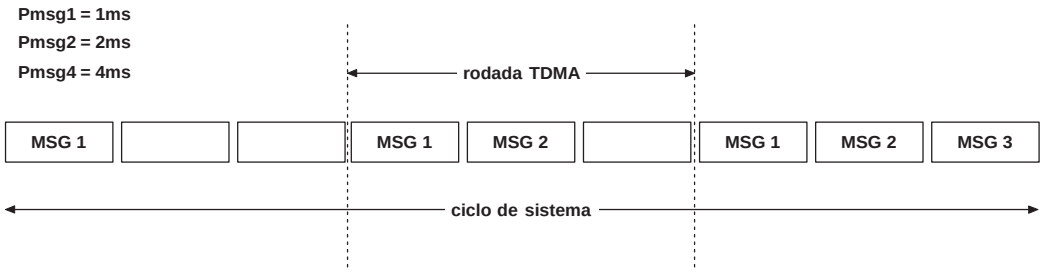

Figura 2.2: Exemplo de comunicação TDMA.

Um exemplo de comunicação em TDMA é ilustrado na Figura 2.2, onde 3 nodos transmitem mensagens com periodicidades de 1, 2 e 4ms, respectivamente. Define-se um horizonte de tempo finito chamado de **ciclo de sistema** sobre o qual todas as transmissões são agendadas para acontecer de forma exclusiva. Assim garante-se que após a integração de 2 ou mais subsistemas o comportamento da comunicação permanece o mesmo previsto quando foram projetados separadamente. A latência de transmissão de todas as mensagens permanece fixa com jitter praticamente zero, pois esta não é alterada pelo aumento ou diminuição da carga na rede. Maiores detalhes sobre a configuração e funcionamento de protocolos TDMA serão apresentados no capítulo 4.

2.3 Tolerância à Falhas em Sistemas Distribuídos

O projeto de sistemas não-críticos, como por exemplo produtos eletrônicos portáteis de alto consumo, é finalizado quando o procedimento de validação funcional adotado

aponta que os requisitos presentes na especificação foram atendidos pela implementação, sem maiores preocupações com a ocorrência de falhas em tempo de operação. Na prática entretanto existe sempre uma probabilidade não nula de que falhas causadas por agentes externos ou internos venham a ocorrer colocando em risco o funcionamento de componentes de hardware e software que compõem o sistema alvo. Particularmente em aplicações críticas, são exigidos elevados índices de confiabilidade, ou seja, estimativas que expressam o número de falhas por unidade de tempo, e disponibilidade, que indica por quanto tempo espera-se que um sistema opere corretamente sem apresentar nenhum comportamento defeituoso. Aspectos não-funcionais se somam aos funcionais para que se possa avaliar a qualidade da solução final, em termos de robustez e segurança. No caso de sistemas embarcados por-fio, é preciso garantir que a ocorrência de erros durante a operação seja menor, ou pelo menos igual, àquelas dos sistemas hidráulicos ou mecânicos que foram substituídos. Por isso, em situações como essas os responsáveis pelo projeto são forçados a considerar o uso de tolerância à falhas na medida em que os esforços necessários para evitar a ocorrência de erros em tempo de execução, seja pelo uso de sistemas de proteção ou pela utilização de componentes mais robustos, se tornam demasiadamente custosos. O projeto de sistemas críticos é de fato uma combinação de prevenção, técnicas de mascaramento de falhas e práticas de projeto tudo no objetivo de diminuir ao máximo o risco de acidentes que possam colocar em risco vidas humanas ou propriedade de alto valor.

2.3.1 Origem do Conceito

O conceito de tolerância à falhas surgiu pela primeira vez na literatura em (AVIZIENIS, 1967), em resposta à crescente necessidade de se construir sistemas cada vez mais robustos que se observava na época.

Originalmente o conceito foi expresso da seguinte forma:

> "*We say that a system is fault-tolerant if its programs can be properly executed despite the occurrence of logic faults.*"
>
> Algirdas Antanas Avizienis

Segundo Avizienis (AVIZIENIS, 1995), a preocupação inicial era com falhas permanentes e transientes, causadas por fenômenos físicos, em sistemas híbridos (hardware & software). Com o aumento da complexidade dos circuitos integrados e dos softwares desenvolvidos, falhas humanas introduzidas em tempo de projeto se tornaram cada vez mais freqüentes e mais difíceis de serem completamente eliminadas durante as fases de verificação funcional e teste estrutural. Como conseqüência, mais tarde estas falhas de natureza humana também foram trazidas para dentro do conceito. Isto se deve sobretudo em virtude do aumento da complexidade de projeto de circuitos integrados e de software.

A evolução do conceito de tolerância à falhas introduziu novos paradigmas de projeto, além de metodologias e normas a serem aplicadas **em todas as fases do ciclo de vida de um sistema**, inclusive na definição da plataforma sobre a qual os requisitos funcionais são mapeados.

2.3.2 Taxonomia Básica

O termo **falha** é muitas vezes usado de maneira inadequada, sendo entendido como um comportamento incompatível no sistema. Contudo, deve-se tomar algum cuidado com a utilização deste termo para que não surjam interpretações ambíguas. Conforme

esclarecido em (PANKAJ, 1994), o conceito de falha é utilizado para classificar fenômenos indesejáveis que podem ou não comprometer a integridade de um sistema. Bugs de software são exemplos de falhas que podem até mesmo nunca se manifestar, apesar de existirem.

A manifestação de uma falha recebe o nome de **erro**. Sistemas tolerantes à falhas devem ser capazes de detectar erros e tomar alguma atitude para impedir que estes erros se propaguem para os demais componentes. Quando erros "escapam" e se propagam para outras partes do sistema, é possível que o sistema inteiro, ou parte de sua funcionalidade entre em colapso, caracterizando uma situação de **defeito**. Por outro lado, quando o sistema consegue confinar um erro dentro de um componente, a falha que o gerou pode ser mascarada com uso de técnicas baseadas em redundância, onde um outro componente idêntico, ou que executa a mesma função, toma o lugar da réplica defeituosa.

2.3.3 Definição do Modelo de Falhas

O projeto de sistemas altamente confiáveis tem por objetivo chegar a uma solução final que seja capaz de mascarar uma ou mais falhas de um determinado conjunto, o qual foi adotado como sendo aquele que melhor representa o que acontece na prática, quando o sistema está em operação. Cabe aos responsáveis pelo projeto avaliar quais são as vulnerabilidades do sistema assim como os possíveis fenômenos adversos provenientes do ambiente externo que podem atingí-lo, considerando as diferentes situações de uso. Experiência de projetos anteriores também é de extrema relevância. Falhas que ocorreram em versões anteriores do sistema são reportadas e incluídas no conjunto de falhas que será reutilizado em projetos futuros.

Falhas podem afetar um sistema computacional nas dimensões de espaço, tempo e valor. Em sistemas distribuídos uma falha na dimensão de espaço é por exemplo a destruição de um nodo, ruptura do barramento, etc. Na dimensão do tempo, a manifestação de uma falha pode ser o atraso ou adiantamento de uma ação ou resultado de uma computação em relação ao instante esperado. Na dimensão de valor, falhas podem fazer com que componentes forneçam resultados inválidos.

Nas dimensões de tempo e valor, falhas devem ser classificadas em função da maneira pela qual o erro gerado é percebido por outros componentes que formam o sistema. Na dimensão do espaço, assume-se que o componente destruído não interfere mais no resto do sistema. Para o estudo de falhar nas dimensões de tempo e valor, foi adotada a classificação proposta por Philip Thambidurai e You-Keun Park apresentada em (THAMBIDURAI; PARK, 1988) por ter sido considerada a mais adequada para o propósito do trabalho. Segundo os autores, as falhas podem ser de três tipos:

1. Falhas Arbitrárias (ou Bizantinas)

   Falhas arbitrárias são caracterizadas por um comportamento totalmente irrestrito e sua manifestação pode ou não ser detectada. Quando ocorrem, é possível que um mesmo resultado seja interpretado de maneira ambígua em diferentes componentes de um sistema distribuído.

2. Falhas Simétricas

   Ao contrário de falhas arbitrárias, as falhas simétricas têm um comportamento restrito apesar de que sua ocorrência também pode ou não ser detectada. Quando ocorrem, resultados inconsistentes são fornecidos para diferentes processadores contudo são sempre percebidos da mesma maneira.

3. Falhas Manifestas

   Falhas manifestas são aquelas cuja ocorrência é sempre detectada por todos os nodos receptores.

Para toda falha existe uma taxa de ocorrência λ associada, a qual indica quantas vezes estima-se que esta falha ocorra em um horizonte de tempo finito. Em (BOWLES, 1992) o autor apresenta uma visão geral dos métodos mais significativos de estimativa de ocorrência de falhas em dispositivos eletrônicos integrados, incluindo métodos utilizados pelo CNET (*Centre National d'Études des Télécommunications*) e Siemens. Estes métodos por sua vez levam em conta fatores tais como:

- Qualidade do produto, avaliado por níveis de inspeção e teste pós manufatura.
- Complexidade do dispositivo, na maioria dos casos uma função do número de células lógicas, bits de memória, etc.
- Tipo de encapsulamento e número total de pinos.
- Temperatura (fixa) de operação.
- Níveis de sobretensão aplicado no dispositivo.
- Condições do ambiente externo.
- Fator de aprendizagem, uma vez que os métodos consideram que quanto mais antigo o produto ou tecnologia menor a probabilidade que falhas ocorram.

Interessante notar que métodos de estimativa de ocorrência de falhas são essencialmente empíricos e fornecem resultados diferentes quando aplicados na avaliação de um mesmo dispositivo. O artigo aplica métodos distintos para estimar o número de falhas por unidade de tempo de uma memória DRAM de 64Kbits, obtendo diferenças de até 150% nos resultados.

Assume-se que o conjunto de falhas potenciais, a correta classificação destas falhas em função do domínio de ocorrência, comportamento assim como as respectivas taxas de ocorrência definem o chamado **modelo de falhas** do sistema. Os responsáveis pela definição do modelo podem ainda considerar que duas ou mais falhas ocorrem ao mesmo tempo, ou que cada falha ocorre isoladamente, o que irá determinar mais tarde o nível de redundância exigido nas técnicas de mascaramento. Em se tratando de aplicações críticas, onde altos níveis de confiabilidade e disponibilidade são exigidos, o modelo de falhas irá influenciar diretamente em todas as fases de um projeto.

Para fins de projeto de sistemas distribuídos, é comum em muitos trabalhos assumir que falhas ocorrem somente em nível de nodo logo, a replicação de uma unidade consiste na inclusão de um novo nodo na rede. Por outro lado, modelos mais realistas consideram que falhas também podem atingir o canal de comunicação, o qual também deve ser replicado neste caso. A replicação do barramento de rede exige que, pelo menos, a camada física e parte da camada de enlace do protocolo seja também replicada nos nodos que fazem uso de mais de um canal para troca de mensagens redundantes.

2.3.4 Técnicas de Redundância

Conforme destaca Avizienis (AVIZIENIS, 1995), tolerância à falhas precisa ser considerada meta fundamental no projeto de aplicações críticas, onde existe um envolvimento direto com a segurança de seres humanos ou de propriedade de alto valor agregado. Como já mencionado, o projeto de alguns sistemas, como é o caso de produtos eletrônicos portáveis, não se preocupa com questões de tolerância à falhas visto que os níveis de confiabilidade e disponibilidade de serviços exigidos são dezenas, senão centenas de vezes inferiores à probabilidade de falhas nos componentes de hardware e software utilizados na sua implementação. Para sistemas críticos portanto a realidade é outra. Em aplicações aeroespaciais, por exemplo, são exigidas taxas de erros iguais ou menores que 10^{-9} por hora (RUSHBY, 2001). Neste mesmo artigo o autor ainda salienta que, apesar de aplicações automotivas exigirem taxas de erros menores, estas são executadas em ambientes mais hostis, dentro dos quais o sistema está sujeito a uma maior intensidade de perturbações externas tais como interferência eletromagnética, sobre-aquecimento, fortes vibrações, colisões, entre outras.

Níveis de confiabilidade mais elevados exigidos em aplicações críticas podem ser atingidos com o uso de técnicas de redundância (DILGER et al., 2005) as quais se baseiam na idéia de que uma unidade de processamento ou qualquer outro componente pode ser rapidamente substituído por uma réplica caso venha a falhar. Assim, a falha de algum componente pode ser "sutilmente" mascarada sem que a execução do sistema em questão seja interrompida. Para que esta substituição (ou **chaveamento**) a quente entre elemento defeituoso e sua respectiva réplica seja feito de forma automática e transparente para o objeto ou processo controlado, ambos devem estar sincronizados nas dimensões de tempo e valor, executando tarefas em perfeita sincronia. O alinhamento temporal das unidades replicadas garante que todos componentes envolvidos estejam de acordo sobre instantes de ativação, cumprimento ou perda de deadlines. Além disso, relações de precedência são mantidas pela observação de uma linha de tempo lógico.

O controle de operação de unidades redundantes deve ser preferencialmente contido dentro da plataforma de comunicação, de forma a separar o máximo possível o desenvolvimento da aplicação dos mecanismos de tolerância à falhas utilizados. Algumas plataformas comerciais já incluem processamento de informações redundantes e controle do estado de operação de réplicas interno aos controladores de rede, como é o caso do protocolo TTP/C (TTP/C, 2003).

Diferentes técnicas de redundância podem ser adotadas no intuito de aumentar a confiabilidade do sistema, não pela diminuição da ocorrência de falhas mas sim pelo seu mascaramento, evitando uma situação de defeito. Contudo, o uso de uma técnica de redundância é direcionado a um modo de falhas, isto é, um subconjunto específico do modelo de falhas do projeto.

2.3.4.1 Unidades de Reserva

Unidades de reserva são nodos de um sistema distribuído executados em segundo plano que têm a função de tomar o lugar de uma réplica operacional que porventura seja atingida por uma falha no domínio do espaço, ou seja, alguma falha permanente que destrua o componente, ou falhas transientes, que atingem o componente por um período de tempo finito. No caso de falhas no domínio do tempo ou valor, é preciso garantir que por definição a unidade que falha apresente um comportamento silencioso, caso contrário este pode "atrapalhar" a unidade de reserva.

A unidade que executa em primeiro plano e a de reserva, também chamada de **sombra**, podem ou não serem idênticas contudo, no primeiro caso é necessário assumir que ambas estão livres de falhas de projeto para excluir a hipótese de que uma mesma falha aconteça ao mesmo tempo em ambas.

Em caso de falha, a unidade defeituosa deve ser rapidamente diagnosticada e isolada do resto do sistema com o uso de mecanismos de detecção de erros tais como time-outs, onde considera-se que uma falha ocorreu caso a unidade não responda após uma certa quantidade de tempo. No caso de sistemas distribuídos com relógios sincronizados, o tempo de time-out pode ser especificado com alta precisão, permitindo que a unidade seja substituída tão rápido que o efeito do chaveamento pode nem ser percebido pela camada da aplicação. Obviamente, a técnica de unidades de reserva não se restringe ao uso de somente duas réplicas contudo, este é o caso mais freqüente.

2.3.4.2 Redundância Ativa

A técnica de redundância ativa se baseia no projeto de unidades idênticas, que executam suas tarefas em total sincronia, sobre uma mesma base de tempo global. Em qualquer instante de tempo todas as réplicas têm o mesmo estado de operação, inclusive o mesmo valor de relógio dentro de certos limites de precisão. Esta técnica é muito semelhante à de unidades de reserva. A diferença básica repousa no fato de que em redundância ativa, todas as réplicas envolvidas executam em primeiro plano, isto é, consomem e produzem dados que são utilizados em outras partes do sistema.

O número de componentes redundantes **N** necessário para mascarar **M** falhas que ocorrem **simultaneamente** depende do comportamento da falha, conforme mostra a tabela 2.1, onde observa-se que um menor número de réplicas é exigido quando o modelo inclui somente falhas manifestas. Logo, é preferível construir o sistema de tal forma que falhas simétricas e bizantinas sejam excluídas do modelo por definição. Isto significa dizer que para uma melhor economia de recursos cada nodo do sistema precisa ser projetado de maneira que qualquer falha seja facilmente diagnosticada pelos demais. Para que isto seja possível, mecanismos de detecção de erros no próprio nodo são implementados no objetivo de paralisar toda e qualquer atividade de comunicação ou simplesmente desconectá-lo do barramento. Como resultado, qualquer que seja falha o nodo assume um comportamento silencioso (*fail-silent*), que pode ser facilmente detectado e interpretado como um erro pelos demais componentes do sistema.

Tabela 2.1: Avaliação do fator de redundância.

Tipo de Falha	Fator de Redundância
Arbitrária	**N>3M**
Simétrica	**N>2M**
Manifesta	**N>M**

2.3.4.3 Diversidade de Versões

Diversidade de versões é uma técnica utilizada para mascarar falhas de natureza humana, ou seja, falhas que foram introduzidas durante a fase de projeto do sistema e que não foram detectadas até o momento de sua execução. Estas falhas podem produzir erros a qualquer momento ou até mesmo nunca se manifestarem. Na prática, é sabido que elas de

fato emergem, e com bastante freqüência, representando cada vez mais um motivo de preocupação sobretudo pelo crescente aumento da capacidade de integração de transistores em um único circuito integrado. De acordo com Janick Bergeron (BERGERON, 2001), de 50 a 80% do tempo de projeto de um circuito complexo pode ser gasto em verificação funcional, no intuito de descobrir falhas humanas e corrigí-las ainda durante a fase de projeto. Mesmo assim, é impossível garantir com 100% de certeza que um determinado sistema é livre de falhas deste tipo.

A replicação de versões é dependente da implementação. Para que esta técnica seja eficiente, as unidades redundantes devem ter sido projetadas por equipes distintas no objetivo de garantir que uma mesma falha não esteja presente em todas elas. Sendo ainda **N** o número de unidades que executam a mesma função em paralelo, e **M**=1 o número de falhas que ocorrem **ao mesmo tempo**, destacam-se alguns casos especiais. Com **N**=2 não é possível mascarar falhas arbitrárias nem simétricas porém, é possível conter o seu efeito impedindo que o erro associado se propague para outras partes do sistema na forma de mensagens inconsistentes. Em caso de desacordo entre as unidades replicadas, as demais recusam ambos os resultados e tomam alguma atitude de emergência, como por exemplo utilizar o último valor calculado ou executar automaticamente uma parada segura. Com **N**>2, pelo menos uma falha simétrica pode ser mascarada por um mecanismo de votação, localizado no lado dos receptores. Para mascarar falhas arbitrárias, têm-se a condição que **N**>3, e a escolha do valor correto deve ser feito por algoritmos de acordo.

2.3.5 Conceito de Parada Segura

Em sistemas com capacidade de processamento limitada, como por exemplo um controlador lógico programável (PLC), a reação à ocorrência de erros consiste tipicamente no desligamento da tensão de alimentação de todas as saídas, paralisando assim toda a aplicação. Independente da origem do erro, a resposta do sistema é sempre a mesma. Esta metodologia é até hoje bastante utilizada para evitar conseqüências catastróficas contudo, o maior problema é que na maioria dos casos o sistema demora um tempo significativo para fazer a aplicação retornar ao estado que estava imediatamente antes da ocorrência do erro. Em sistemas industriais isto implica em maiores gastos de energia e paradas temporárias na linha de produção ou processo.

Sistemas mais inteligentes, como é o caso de sistemas distribuídos formados por dispositivos com capacidade de processamento própria, podem ser programados de forma que rotinas de emergência mais eficientes sejam executadas em situações de perigo. Deve ser feito um rápido diagnóstico da situação de falha e assim tomar uma decisão mais precisa, dentro de um conjunto maior de possibilidades e não apenas uma, que desliga toda a aplicação. O desligamento pode ser do tipo granular, ou seja, o sistema decide que somente um pequeno conjunto de componentes deve ser desativado, fazendo com que o sistema opere com uma certa degradação de performance. Para isto, serviços diagnóstico mais avançados se tornam necessários, afim de que erros possam ser rapidamente detectados e isolados. Um exemplo destes serviços é a lista de vida, implementada diretamente nos controladores de rede, e que fornece a cada componente o estado de operação dos nodos. Quando algum nodo falha, os demais tomam conhecimento deste evento dentro de um curto espaço de tempo.

2.4 Modelo de Sistema Distribuído Adotado

Assume-se como um sistema distribuído um sistema computacional **síncrono**, formado por unidades de processamento, também chamadas de nodos, que estão fisicamente distribuídas no espaço, sendo que cada nodo possui pelo menos um processador, um oscilador, memória local e controlador de rede. Em cada nodo do sistema, são executadas tarefas de controle e um algoritmo de sincronização de relógios, também chamado de **processo de sincronização**.

A distribuição espacial é geograficamente limitada às dimensões do objeto ou processo controlado com o qual o sistema interage fortemente. Os nodos se comunicam por troca de mensagens transmitidas através de um meio físico serial de difusão. Não são considerados sistemas distribuídos cuja finalidade é interligar grandes sistemas de informação e de usuários, visando aplicações de comércio eletrônico, gestão, entretenimento, entre outros, como é o caso de redes locais e metropolitanas convencionais.

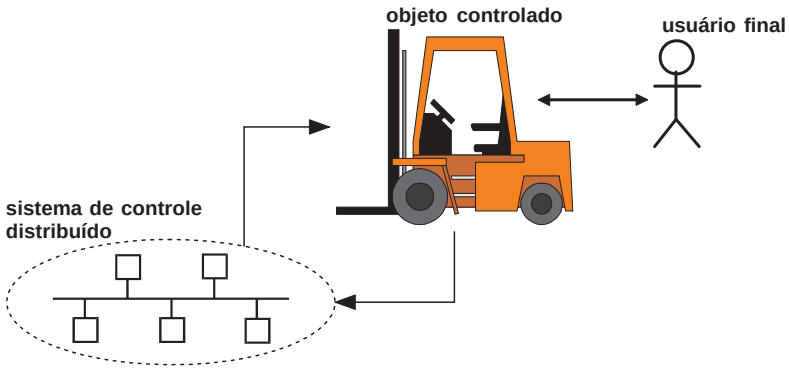

Figura 2.3: Relação entre sistema, objeto controlado e usuário.

A interação com o usuário final é minima ou inexistente, conforme mostra a Figura 2.3. Os componentes de um sistema distribuído embarcado recebem informações de sensores e devolvem para atuadores que irão interferir por sua vez na dinâmica do objeto controlado, caracterizando um comportamento regular, em que as ações do sistema são executadas automaticamente. Esta característica irá permitir excluir falhas humanas do modelo.

Os nodos do sistema se comunicam essencialmente por troca de mensagens através de uma infraestrutura de comunicação que consiste em um canal serial de difusão compartilhado, cujo acesso deve ser arbitrado. O canal serial pode ser tanto um barramento ou uma estrela, conforma ilustra a Figura 2.4. Pontes e acopladores (*star couplers*) intermediários podem existir, lembrando que ponte é um componente de rede que trabalha em nível de enlace.

A pilha de rede do protocolo é formada somente pelas camadas 1, 2 e 7. A camada 7 consiste basicamente em uma área de armazenamento temporário de dados, também chamada de **buffer**, onde as tarefas da aplicação buscam ou depositam novas mensagens para transmissão. Assume-se que a camada da aplicação compreende todas as tarefas de controle dinâmico presentes no sistema distribuído.

Falhas ocorrem tanto a nível de nodo quando a nível de barramento. Muitos trabalhos relacionados ao assunto assumem por conveniência que somente nodos falham porém, sabe-se que na prática existe alta probabilidade de falhas ocorrerem na infraestrutura física de rede, como é o caso de falhas transientes ou em rajada causadas por interferência eletromagnética.

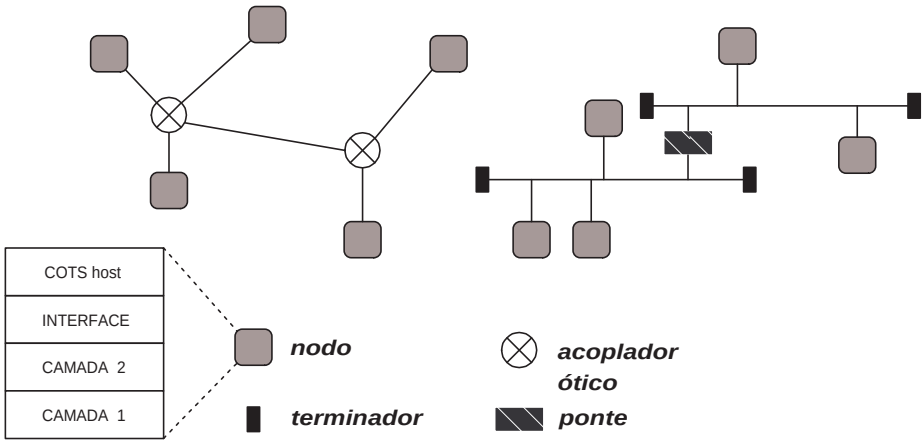

Figura 2.4: Infraestrutura física adotada.

Ao contrário de redes de computadores que têm por finalidade interligar grandes sistemas de informação e pessoas, este trabalho tem seu foco em sistemas distribuídos construídos para solucionar problemas mais específicos, como é o caso de aplicações embarcadas ou industriais. Neste contexto os nodos são sensores e atuadores com capacidade de processamento e comunicação em rede que executam tarefas de controle interagindo diretamente com a dinâmica do ambiente externo.

Um conjunto de nodos que executa uma determinada função também pode ser visto como um subsistema independente dos demais do ponto de vista de projeto, conforme já mencionado no item 2.2.1 onde a questão do particionamento temporal foi discutida. Exemplos de subsistemas em aplicações críticas são malhas de controle, cada uma formada pelo menos por um nodo atuador e um sensor, conforme mostra a Figura 2.5. Logo, um sistema distribuído também pode ser visto como um conjunto de subsistemas que executam funções independentes umas das outras.

Define-se que a plataforma de comunicação consiste na implementação de uma pilha de rede juntamente com toda a infraestrutura física que interliga os componentes de um sistema distribuído, tais como cabos, conectores, repetidores e transdutores de sinais analógicos. Para melhor facilitar a apresentação e a compreensão da proposta deste trabalho, a implementação da pilha será também chamada de **bloco lógico** enquanto que a infraestrutura física será chamada de **bloco físico**.

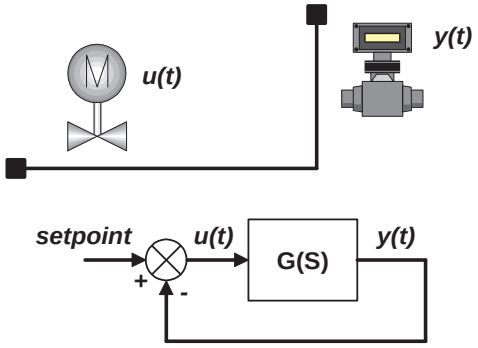

Figura 2.5: Exemplo de subsistema elementar

A estrutura de dados elementar utilizada na comunicação entre as tarefas da camada da aplicação recebe o nome de **mensagem**. Durante a execução do sistema, mensagens

são produzidas ou consumidas pelas tarefas que estão sendo executadas em processadores distintos. Quando uma tarefa deseja transmitir uma mensagem, esta deposita seu conteúdo nos buffers da camada da aplicação para que seja enviada serialmente até um ou mais nodos de destino.

As mensagens trocadas entre as tarefas carregam informações que podem ser alarmes ou variáveis de processo. Alarmes são notificações da ocorrência de eventos externos ou internos enquanto que variáveis de processo são amostras de alguma grandeza física sob controle ou observação do objeto controlado cuja validade no tempo é finita. Mensagens de alarmes devem ser colocadas em fila, tanto na origem (nodo transmissor) quanto no destino (nodo receptor). Por outro lado, mensagens de variáveis de processo são tratadas de forma particular. Elas devem ser constantemente atualizadas pela plataforma para que seu conteúdo não perca a validade. No lado transmissor, cada mensagem produzida sobrescreve a mais antiga presente no buffer de saída. No lado receptor, mensagens de processo são lidas mas não são consumidas, ou seja, seu conteúdo é mantido no buffer de entrada. Quando uma nova mensagem chega, esta automaticamente substitui a mais antiga, tendo ela sido lida ou não.

Na pilha de rede, a estrutura de dados elementar recebe o nome de **quadro** que compreende basicamente um cabeçalho de controle, um campo de dados e um CRC. Toda informação trocada entre as tarefas deve ser encapsulada dentro de um quadro antes da transmissão. Uma mensagem pode ser transmitida em mais de um quadro (fragmentação) assim como um único quadro pode comportar mais de uma mensagem.

Por fim, uma última característica importante que vale mencionar é que o tempo de transmissão dos quadros é na ordem de centenas de vezes maior do que o processamento das tarefas nos processadores.

2.5 Paradigmas de Comunicação

Aplicações distribuídas consistem em tarefas que devem cooperar na execução de uma determinada função utilizando os serviços que a plataforma de comunicação oferece. Tendo em vista que a única forma de comunicação é a troca de mensagens, é importante a definição de um modelo abstrato para orientar o desenvolvimento da camada da aplicação, sobretudo no que se refere à sincronização das ações do sistema. Este modelo recebe o nome de **paradigma de comunicação**, que define basicamente a maneira pela qual a camada da aplicação faz uso dos serviços de rede para coordenar as atividades de tarefas que são executadas em nodos distintos.

Cabe aqui salientar que paradigma de comunicação e controle de acesso não significam a mesma coisa, apesar de que muitas vezes estes dois conceitos estão diretamente relacionados. Tomando como referência o modelo OSI, o primeiro diz respeito a uma estratégia de camada 7 enquanto que o outro consiste num serviço de camada 2. Em casos parcitulares, o paradigma de comunicação está estritamente ligado ao controle de acesso.

Uma breve descrição dos principais paradigmas de comunicação utilizados em aplicações industriais e embarcadas é fornecida a seguir.

2.5.1 Cliente-Servidor

Nodos clientes iniciam a comunicação com um ou mais nodos servidores que têm a função de oferecer um conjunto de serviços específicos. Tem-se que os clientes conhecem os servidores, ou seja, sabem como localizá-los e quais os serviços disponíveis. Por outro lado, nodos servidores não necessariamente conhecem os clientes.

2.5.2 Produtor-Consumidor

Define que existem nodos produtores, que são programados para enviar mensagens em broadcast sempre que alguma nova informação relevante para o resto do sistema é produzida, e nodos consumidores, que são previamente configurados para "filtrar" o tráfego da rede e receber somente aquelas mensagens que lhes são de interesse.

2.5.3 Mestre-Escravo

Define um nodo mestre, que é o único responsável por ordenar todas as ações do sistema, inclusive no que diz respeito ao uso do meio físico para transmissão de dados. Os demais nodos são escravos e transmitem suas mensagens somente quando requisitado pelo mestre. O paradigma mestre-escravo é o único que pode ser implementado também como mecanismo para controle de acesso, sendo este um caso particular onde os dois conceitos se confundem. No entanto, para fins de sincronização de tarefas, o paradigma mestre-escravo pode ser utilizado sobre qualquer outro protocolo, independente da estratégia de controle de acesso.

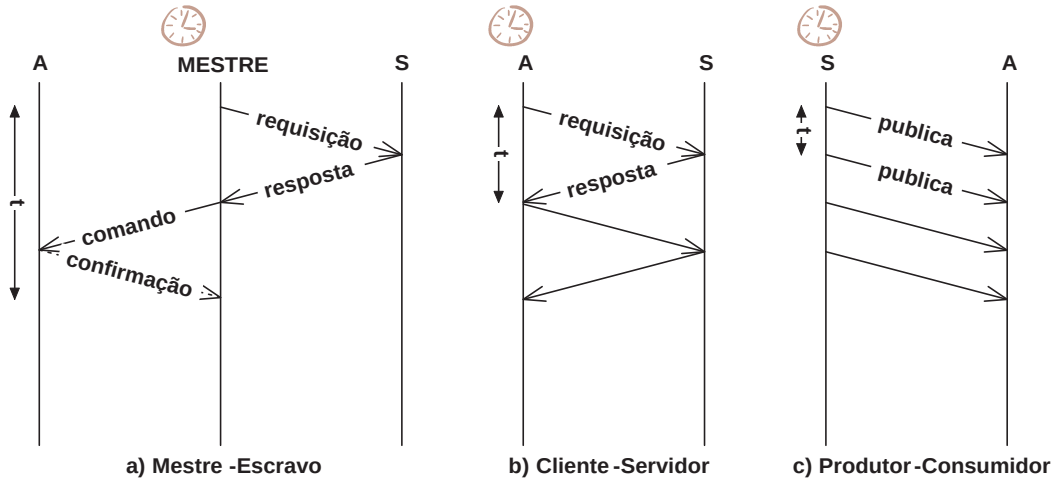

Figura 2.6: Diferentes modelos de comunicação.

Em se tratando de aplicações de controle, interessa poder relacionar os diferentes paradigmas com a questão do tempo. A Figura 2.6 ilustra os três possíveis padrões de comunicação de um subsistema que deve executar uma tarefa elementar que consiste basicamente em três passos (leitura-cálculo-escrita) que são executados a cada ciclo de controle. A orientação vertical indica a evolução do tempo, no sentido de cima para baixo. A leitura é realizada em um nodo do tipo S (sensor), a escrita em um nodo tipo A (atuador) e o cálculo em S, A, ou qualquer outro nodo com capacidade de processamento. O símbolo de relógio indica onde a referência de tempo da tarefa está localizada, assumindo inicialmente que as bases de tempo não estão sincronizadas. Observa-se que no modelo mestre-escravo é preciso transmitir no mínimo 3 quadros para executar um ciclo da tarefa. Em alguns casos, a última mensagem enviada requer uma confirmação, aumentando para 4 o número mínimo de quadros. No modelo cliente-servidor, é possível fazer do nodo atuador um cliente diminuindo o número mínimo de quadros para 2, ao passo que no modelo produtor-consumidor é possível executar o ciclo básico de controle transmitindo somente uma mensagem.

Os três exemplos mostrados na Figura 2.6 mostram casos ideais, onde cada quadro é transmitido imediatamente sempre quando solicitado pela camada da aplicação. Na

prática, um mesmo sistema distribuído pode ter vários clientes ou produtores logo, considerando que as bases de tempo não estão sincronizadas, colisões de acesso ao meio físico podem ocorrer e retardar o envio de cada uma das mensagens.

3 SINCRONIZAÇÃO DE RELÓGIOS

3.1 Apresentação Informal do Problema

Em um sistema distribuído, onde várias tarefas devem cooperar no fornecimento de um determinado serviço, é fundamental o uso de mecanismos de sincronização para garantir que um comportamento coerente ao longo do tempo seja mantido. Em se tratando de sistemas distribuídos de tempo-real, esta cooperação implica em restrições de:

TEMPO

: representando basicamente nos instantes de ativação e os prazos limites para término (*deadlines*) das tarefas do sistema e

CAUSALIDADE

: que expressam uma seqüencia predeterminada de ações que deve ser respeitada.

Quando as restrições de tempo e/ou causalidade são corrompidas, existe o risco de que o sistema entre em colapso colocando em risco a segurança de seus usuários. Em uma malha de controle digital, por exemplo, uma ação de escrita de um valor em um dispositivo atuador deve obrigatoriamente ser precedida do cálculo deste valor. Um sistema de controle de vôo deve autorizar a aterrissagem da aeronave somente após a abertura do trem de pouso ter sido efetuada com sucesso.

Sistemas distribuídos também podem ser construídos com a finalidade de capturar eventos externos com instantes de ocorrência imprevisíveis. No momento em que são percebidos pelo sistema, os eventos provenientes de diferentes fontes devem ser devidamente ordenados para que se possa inferir conclusões corretas sobre o comportamento de um fenômeno[1] qualquer ocorrido no ambiente. Para isto, os registros destes eventos devem ser expressos não só em função de sua origem como também do instante de ocorrência. Considerando que, eventos podem ser percebidos por diferentes elementos que estão fisicamente distribuídos, a existência de uma base de tempo global permite que para cada evento seja associado um valor de tempo lógico. Desta forma, em qualquer ponto do sistema, a interpretação de um dado fenômeno será exatamente a mesma, tanto no domínio do espaço quanto no domínio do tempo.

Por definição, os elementos de processamento de um sistema distribuído estão fisicamente separados no espaço, cada um com o seu próprio conjunto de recursos de hardware, tais como memória, processador, relógio, etc, que não são diretamente acessíveis por tarefas executadas em outros nodos. Além disso, na grande maioria dos casos o sistema é

[1] Entende-se por fenômeno um conjunto de eventos que ocorrem em um intervalo de tempo limitado.

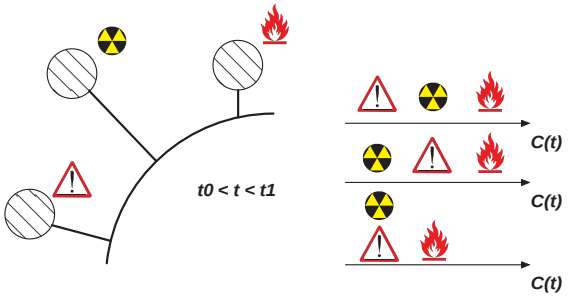

Figura 3.1: Ordenamento de eventos externos.

heterogêneo, ou seja, a configuração de hardware e software não é a mesma para cada nodo. Tarefas distribuídas são executadas em plataformas com recursos distintos e independentes, o que torna impossível uma definição global homogênea dos tempos de processamento das tarefas caso não exista nenhum mecanismo de sincronização disponível. Em princípio, cada nodo possui um relógio local independente e um processador operando a uma determinada freqüência, além de dispositivos de entrada e saída e chamadas de sistema que apresentam diferentes tempos de resposta.

No caso de relógios, ainda que a freqüência da fonte primária fosse a mesma para todos os nodos, sabe-se que na prática os dispositivos osciladores possuem sempre uma incerteza sobre seus valores nominais de freqüência, geralmente expressa em partes por milhão (ppm). Este pequeno erro faz com que relógios distribuídos divirjam cada vez mais ao longo do tempo. Cristais osciladores, por exemplo, podem apresentar um erro de até 10 ppm, ou $10^{-5} s.s^{-1}$ (SURI; HUGUE; WALTER, 1994). Considerando este erro ao longo de um dia, os relógios de dois nodos podem diferir de até 1.72 segundos no pior caso. Logo, mesmo se dois nodos fossem inicializados no mesmo instante de tempo em plataformas idênticas, com o mesmo processador, mesma configuração de memória e mesma freqüência nominal de operação, ainda assim seria impossível que um deles pudesse inferir com precisão qualquer informação sobre o estado de execução do outro somente a partir de informações locais. A imperfeição dos dispositivos osciladores é a principal razão pela qual nenhuma relação consistente entre instantes de início e término de tarefas distribuídas pode ser estabelecida, caso os relógios não sejam submetidos a correções regulares.

3.1.1 Sincronização por Eventos

Uma vez que tarefas se comunicam apenas por troca de mensagens, a única maneira de se obter informações sobre o estado de execução de tarefas não-locais é observar a ocorrência de eventos na rede e tratar o conteúdo das mensagens recebidas. Soluções para o problema de sincronização em sistemas distribuídos relacionadas em (SURI; HUGUE; WALTER, 1994) são classificadas de duas formas, a primeira baseada em eventos, chamada de **sincronização implícita**, e a segunda baseada no estabelecimento de uma base de tempo comum através de técnicas de sincronização de relógios, chamada de **sincronização explícita**.

Na sincronização por eventos, o que acontece basicamente são paradas estratégicas no fluxo de execução de uma tarefa para esperar uma resposta de outro nodo através do envio de uma mensagem. Desta forma, é possível ordenar as ações do sistema somente com base em eventos ocorridos na rede. Exemplos de paradas são primitivas de comunicação do tipo *send()* e *receive()* bloqueantes que são explicitamente inseridas no código

da aplicação. Para evitar que uma tarefa qualquer fique aguardando indefinidamente por uma mensagem que foi destruída em razão de falhas na rede, faz-se uso de primitivas de tempo locais para resumir a parada e dar continuidade à execução do programa após uma determinada quantidade de tempo, também chamada de *timeout*. O tempo de *timeout* em cada parada é uma estimativa que depende de um conhecimento prévio do tempo de processamento no nodo do qual se espera a resposta e dos atrasos de transmissão através da rede (LAMPORT, 1984).

Infelizmente, a avaliação dos atrasos da rede é muitas vezes dificultada, principalmente em sistemas construídos sobre infraestruturas de comunicação mais irregulares, como no caso de redes ad hoc ou arquiteturas altamente fragmentadas compostas de várias subredes. Nestes casos, atrasos de transmissão das mensagens sofrem maiores variações além do aumento significativo do risco de perda ou duplicação de dados. Em razão disso, torna-se necessária a implementação de serviços de sincronização por eventos de mais alto nível sobre da pilha de comunicação, como por exemplo serviços de broadcast atômico (DEFAGO; SCHIPER; URBAN, 2004) os quais prevêem diversas situações adversas.

A coordenação em sistemas que utilizam sincronização implícita é dita relativa, uma vez que as relações de precedência são expressas sempre em função da ocorrência de eventos sem nenhuma relação direta com a linha do tempo. Apesar de possibilitar um correto seqüenciamento de ações, soluções deste tipo se mostram ineficientes para aplicações de tempo-real, as quais interagem fortemente com o ambiente externo. Em sistemas onde existe um envolvimento direto com a dinâmica do ambiente sob controle, é fundamental que exista um mapeamento entre o tempo físico e a seqüencia lógica dos acontecimentos internos. Isto porque muitas das ações executadas por estes sistemas são leituras e escritas de variáveis de processo associadas a entidades físicas com requisitos temporais, incluindo processamento em tempo real de resultados envolvendo estas variáveis.

3.1.2 Sincronização por Correção de Relógios

A sincronização por eventos possibilita apenas um correto ordenamento de acontecimentos mais relevantes sem nenhuma relação direta deste ordenamento com o tempo físico transcorrido. Por outro lado, técnicas de sincronização de relógios trabalham explicitamente com a noção de tempo. Cada componente do sistema possui um relógio local que representa um entendimento global do tempo e que a cada momento pode ser consultado. Além disso, é possível associar o valor discreto fornecido por este relógio com instantes ou intervalos de tempo físico.

A sincronização de relógios tem como objetivo manter uma base de tempo comum, visível por todos os seus componentes para que estes possam coordenar suas atividades respeitando as restrições de tempo e causalidade mencionadas anteriormente. Seja qual for o método aplicado, as principais etapas do processo de sincronização de relógios, também chamado de **algoritmo de sincronização**, são as seguintes:

1. Cada processo informa o valor de seu relógio local a outro(s) processo(s).

2. A partir do(s) valor(es) informado(s) por outro(s) processo(s), cada processo calcula um respectivo fator de correção que expressa o desvio entre a sua base de tempo e uma referência global.

3. Cada processo aplica o fator de correção no seu relógio local.

No momento em que os relógios locais são constantemente corrigidos em função de leituras de relógios de outros nodos, a base de tempo local deste componente deixa de ser

independente e passa a ser um entendimento comum para todo o sistema.

Em se tratando de sistemas distribuídos que executam funções críticas, mecanismos de tolerância à falhas são implementados sobre relógios sincronizados para que as restrições de tempo e causalidade sejam respeitadas mesmo em condições extremas, como no caso de um de seus componentes assumir um comportamento totalmente arbitrário. Contudo, para garantir que a solução final seja resiliente, o algoritmo de sincronização deve ser ele mesmo tolerante à falhas e oferecer níveis de confiabilidade e disponibilidade maiores ou, no pior caso, iguais aos dos mecanismos de detecção de erros e mascaramento de falhas que utilizam seus serviços.

Diferentes técnicas de sincronização de relógios podem ser classificadas de acordo com a forma de implementação. Em um extremo, existem implementações em baixo nível que solucionam o problema em hardware aplicando correções em tempo de ciclo de máquina diretamente nos dispositivos físicos. Por outro lado, soluções em alto nível fazem uso de algoritmos de convergência ou de consistência em software que atuam sobre relógios lógicos para o estabelecimento de uma visão de tempo comum.

3.2 Formalização do Problema

Assume-se que cada nodo do sistema possui um relógio lógico C, o qual possibilita a associação de cada instante de tempo físico t a um valor lógico $T = C(t)$. Define-se também uma função inversa $c = C^{-1}$ que associa cada T a um instante de tempo físico $t = c(T)$. Por conformidade, no decorrer do texto o uso de caracteres ocidentais em caixa alta indicará valores lógicos de tempo, enquanto que caracteres em caixa baixa serão usados para expressar algum instante ou intervalo sobre a linha de tempo físico.

O tempo físico t, também chamado de tempo absoluto, consiste na linha de tempo Newtoniana que rege a ordem cronológica de todos os acontecimentos do mundo real, enquanto que o tempo lógico é um valor constantemente atualizado em intervalos discretos mas que pode ser efetivamente utilizado na computação.

Relógios lógicos são funções discretas pois na prática são implementados na forma de circuitos contadores incrementados a cada pulso de máquina, ou pelo estouro de um outro contador, que por sua vez é incrementado por pulsos de máquina, e assim por diante. Em função do valor nominal da freqüência do oscilador utilizado como fonte primária de tempo, é feita uma associação direta entre o estado do contador com um valor de tempo lógico. Um relógio lógico implementado como um contador sendo incrementado a cada pulso de cristal com freqüência de 1Mhz, fornece a cada instante de tempo físico t um valor $T = (10^{-6}Z)$ segundos, onde Z é o número binário que indica o número de pulsos de máquina contados em um dado momento.

Supondo a existência de um relógio perfeito, T segundos de tempo lógico corresponderiam exatamente a t segundos de tempo físico, isto é, a taxa de atualização teria precisão absoluta em relação a linha de tempo físico, o equivalente a dizer que:

$$\frac{dC(t)}{dt} = \frac{dT}{dt} = 1 \tag{3.1}$$

Infelizmente, a relação 3.1 não representa o que acontece na prática pelo simples fato de ser fisicamente impossível construir um oscilador perfeito cuja freqüência seja exata. Seja qual for a fonte primária de tempo utilizada, sua freqüência de operação é expressa por um valor nominal e uma incerteza em cima deste valor. Isto significa que, a freqüência de operação real efetiva pode assumir qualquer valor dentro de uma pequena

faixa cujo centro coincide com o ponto de freqüência nominal. Em razão disto, torna-se necessária a definição de um relógio real e sua relação com o tempo físico. De acordo com Schneider (SCHNEIDER, 1986), um relógio em um sistema distribuído é assumido como não-defeituoso se a seguinte condição for satisfeita:

$$1 - \rho \leq \frac{C(t_2) - C(t_1)}{t_2 - t_1} \leq 1 + \rho \qquad (3.2)$$
$$\forall \quad t_2 > t_1$$

Na relação 3.2, ρ é a incerteza do oscilador, que representa o desvio máximo da freqüência real em relação a freqüência nominal de um relógio C. Visto de outra forma, ρ pode ser interpretado como o quão rápido o tempo lógico se distancia do tempo absoluto durante regime normal de operação do sistema.

É importante observar que a magnitude do desvio entre tempo lógico e tempo físico durante um certo intervalo de tempo é limitada a $\rho\Delta t$, onde $\Delta t = t_2 - t_1$. No entanto, a magnitude máxima do desvio entre dois relógios lógicos independentes pode ser de até $2\rho\Delta t$ considerando o pior caso em que durante Δt segundos um deles desvia $+\rho\Delta t$ e o outro $-\rho\Delta t$.

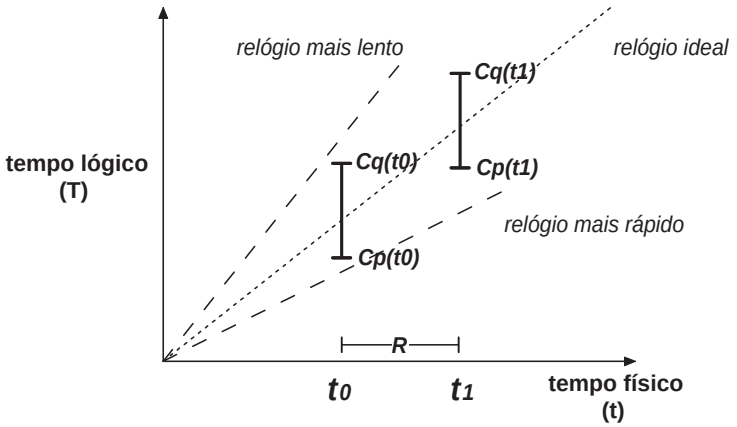

Figura 3.2: Relação entre tempo lógico e tempo físico.

A Figura 3.2 ilustra a evolução de relógios lógicos em função do tempo físico transcorrido destacando o efeito de ρ quando não existe nenhum mecanismo de sincronização. O relógio mais rápido opera a uma freqüência efetiva igual a $(1 + \rho)f$ enquanto que o mais lento opera a $(1 - \rho)f$, onde f é a freqüência nominal da fonte primária de tempo. Ainda na Figura 3.2, C_p e C_q representam dois relógios sincronizados. Neste caso, a qualquer instante de tempo físico seus relógios lógicos diferem porém, a diferença máxima é limitada a um certo valor.

Tomando como referência o modelo apresentado em (SRIKANTH; TOUEG, 1987), as condições para o estabelecimento de uma base de tempo global são as seguintes:

PRECISÃO

Implica que dois relógios lógicos quaisquer, não-defeituosos, se mantém alinhados dentro de certos limites de precisão. Sendo p e q processos que executam um algoritmo de sincronização e Cp e Cq seus respectivos relógios, então em qualquer instante de tempo físico t:

$$|C_p(t) - C_q(t)| \leq \delta \qquad (3.3)$$

onde δ é o desvio máximo de tempo existente entre p e q. Quanto menor o valor de δ, mais perfeito será a associação entre tempo lógico e tempo físico e a base de tempo global terá uma granularidade mais fina, permitindo assim ordenar eventos cada vez mais próximos um do outro e programar ações em instantes de tempo mais precisos.

Apesar de ser menos intuitiva, uma outra forma de expressar a condição de precisão é a seguinte:

$$|c_p(T) - c_q(T)| \leq \hat{\delta} \tag{3.4}$$

onde $\hat{\delta}$ é o desvio observado em função do tempo físico. A relação entre δ e $\hat{\delta}$ é a mesma existente entre qualquer unidade de tempo lógico e tempo físico.

EXATIDÃO

Implica que qualquer relógio lógico de uma base de tempo global se mantém suficientemente próximo a uma referência de tempo absoluta. Sendo p um processo qualquer e Cp seu respectivo relógio, então em qualquer instante de tempo físico t a condição de exatidão é satisfeita se:

$$|C_p(t) - (t)| \leq \gamma \tag{3.5}$$

onde γ é o máximo desvio de T em relação ao tempo físico t.

A condição de precisão é essencial ao passo que a condição de exatidão se torna necessária somente quando o sistema deve interagir com outros sistemas distantes, ou quando o diâmetro de atuação do sistema em questão extrapola limites de cidades e países. Quando um sistema atende somente a condição de 3.3 precisão então é dito que este sistema possui sincronização interna. No caso de sistemas que atendem a ambas as condições de precisão e exatidão, a sincronização é do tipo externa.

No domínio de aplicações de controle embarcadas e industriais, a sincronização interna é suficiente para coordenar a execução de tarefas com requisitos de tempo-real. O mais importante é preservar relações de precedência e manter a regularidade das ações de natureza periódica, isto é, o intervalo de tempo entre duas execuções de uma mesma ação não deve sofrer variações. O comportamento de sistemas de controle costuma ser definido dentro de um horizonte de tempo finito, onde ao final da execução da última tarefa programada neste horizonte de tempo, o sistema reinicia a contagem do tempo executando todas as tarefas mais uma vez, e assim sucessivamente, caracterizando um comportamento cíclico. Não existe, a princípio, a necessidade de consultar uma base de tempo absoluta.

Sincronização interna pode ser atingida com a execução de um algoritmo de sincronização de relógios lógicos, corrigidos unicamente em função da leitura de outros relógios presentes no sistema. Por outro lado, a implementação de sistemas com sincronização externa depende do acesso a uma base absoluta, isto é, uma referência de tempo universal como por exemplo o UTC (Coordinated Universal Time). Uma solução para redes altamente dispersas que necessitam de sincronização externa é o NTP (Network Time Protocol) descrito em (MILLS, 1991).

3.3 Sincronização de Relógios em Hardware

Como já mencionado, mecanismos de sincronização de relógios podem ser implementados em diferentes níveis da arquitetura de um sistema distribuído. Implementações

em mais baixo nível são possíveis, nas quais os relógios são sincronizados em tempo de ciclo de máquina.

A Figura 3.3 mostra o esquema simplificado de um circuito de sincronização de i entradas, onde cada entrada representa o sinal de saída do circuito de sincronização de um outro nodo, trazido por um caminho físico dedicado. O seletor do sinal de referência escolhe um dos sinais de acordo com um critério pré-estabelecido, tomando como parâmetro de escolha o desvio de fase cada um. Os blocos VCO (Voltage Controlled Oscillator) e o detector de fase formam um circuito PPL (Phase-Locked Loop) que tem a função de fazer com que o sinal de relógio local coincida com o sinal de referência selecionado. Conforme descrito em (RAMANATHAN; SHIN; BUTLER, 1990), de acordo com o critério de escolha do sinal de referência, o circuito da Figura 3.3 pode tolerar até m falhas arbitrárias nos sinais de clock, na condição de que o número total de sinais disponíveis seja maior do que $3m$.

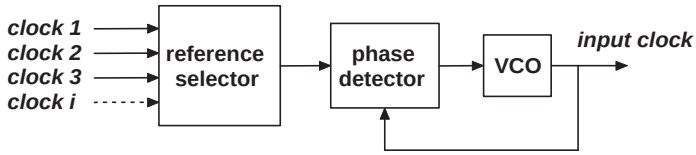

Figura 3.3: Correção de relógios em hardware.

Soluções em hardware têm a vantagem de alcançar maiores níveis de precisão, na ordem de nano-segundos. Isto significa que a granularidade da base de tempo global do sistema distribuído será equivalente àquela de um sistema centralizado, que é limitada apenas pela incerteza sobre a freqüência nominal dos osciladores. No exemplo mostrado na Figura 3.3, a única fonte de erros na leitura da fase dos relógios distribuídos é a variação dos atrasos de propagação de sinal. Outra característica importante é que a correção de relógios é executada de maneira contínua, em paralelo com o processamento do software na plataforma, sem interferir em nenhum outro serviço ou função do sistema. Todavia, o custo de implementação associado é alto pela necessidade de recursos de hardware adicionais, como por exemplo uma rede de interconexão adicional para cada um dos sinais de oscilador além de circuitos PLL dedicados.

3.4 Sincronização de Relógios em Software

Ao contrário de soluções em baixo nível, a sincronização de relógios por software é mais flexível e menos custosa. Mesmo que não seja possível alcançar os mesmos níveis de precisão atingidos por implementações dedicadas em hardware, as soluções em software têm a vantagem de não exigir interconexões extras de sinais nem circuitos de correção de fase dedicados. Toda a troca de informação necessária pode ser feita através do mesmo barramento por onde os dados circundam, além de que o algoritmo pode ser executado no mesmo processador que executa o código da aplicação.

Algoritmos executados em software não interagem diretamente com as camadas de mais baixo nível do protocolo sobre o qual são implementados. As propostas são genéricas e a princípio podem ser implementadas em cima de qualquer plataforma de comunicação, na condição de que atrasos de propagação máximo e mínimo sejam conhecidos para que se possa estabelecer limites sobre o erro máximo associado a cada leitura, conforme será visto mais adiante.

Partindo do princípio que, inicialmente todos os relógios do sistema estão sincronizados, ou seja, seus valores iniciais são aproximadamente os mesmos, o algoritmo de sincronização tem a função de aplicar periodicamente correções em relógios lógicos no objetivo de mantê-los alinhados dentro de certos limites de precisão δ. A criação inicial da base de tempo distribuída é um problema a parte. Em (LONN, 1999a) é feita uma análise comparativa de alguns métodos teóricos de atingir sincronização inicial em sistemas distribuídos destinados a aplicações de controle. Como exemplos práticos, a especificação dos protocolos FlexRay (FLEXRAY, 2004) e TTP/C (TTP/C, 2003) fornecem cada uma a descrição detalha do procedimento de inicialização ressaltando todos os problemas relacionados.

A cada execução do algoritmo de sincronização uma referência de tempo deve ser rapidamente estabelecida entre os nodos por meio de acordo. A partir desta referência, que representa um consenso global naquele instante, todos os processos calculam um fator de correção que será aplicado em seu relógio local para compensar o efeito de ρ. Em regime normal de operação, a referência é sistematicamente reconstruída a partir de leituras de relógios obtidas por meio de solicitações explícitas ou pela coleta de valores espalhados em instantes pre-determinados. Quando a referência de tempo é determinada pela leitura do relógio de um único nodo, então assume-se que o sistema distribuído possui uma base de tempo global **centralizada**. Contudo, a centralização da base de tempo introduz um ponto crítico de falhas no sistema. Caso o nodo de referência caia ou apresente algum comportamento inconsistente, todos os demais nodos perdem a sincronização. Por outro lado, a referência pode ser calculada sempre quando solicitado pelo algoritmo a partir de um conjunto de leituras de relógio, o que caracteriza uma base de tempo descentralizada, ou **essencialmente distribuída**.

A maior vantagem da descentralização da base de tempo está no fato de que eventuais falhas em qualquer um dos componentes do sistema podem ser mascaradas pelo uso de algoritmos de acordo exato ou aproximado, inclusive falhas arbitrárias.

3.4.1 Formas de Acordo

Aparentemente as propostas mais significativas relacionadas a este assunto surgiram em meados da década de 80. Leslie Lamport e Melliar-Smith em (LAMPORT; MELLIAR-SMITH, 1985) propuseram algoritmos de acordo que diferem na maneira pela qual o efeito de falhas em componentes isolados é amenizado. Os autores partem do pressuposto que todos os relógios estão inicialmente sincronizados e que relógios lógicos são atualizados “aproximadamente” na mesma freqüência, isto é, estão todos sujeitos ao efeito de ρ. As duas formas de acordo introduzidas neste trabalho se tornaram um forte embasamento teórico para todas as demais propostas que seguiram até os dias de hoje.

3.4.1.1 Consistência Interativa ou Acordo Exato

O objetivo de algoritmos de acordo exato é fazer com que todos os componentes do sistema acordem sobre um mesmo valor inicializado por uma determinada fonte. A proposta mais conhecida é o algoritmo de mensagens orais, originalmente apresentado em (LAMPORT; SHOSTAK; PEASE, 1982) como solução para o problema dos Generais Bizantinos, enigma clássico da computação onde é feita uma analogia entre processos de um sistema distribuído e generais de um exército que buscam estabelecer acordo sobre um determinado plano de ataque. Cada general e seu respectivo exército está suficientemente distante para assumir que a única forma de comunicação com outro general é através da troca de mensagens. A dificuldade no estabelecimento do acordo está no fato

que alguns generais (processos) podem ser traidores (defeituosos) e assim assumir um comportamento totalmente arbitrário, enviando até mesmo informações conflitantes para diferentes generais no intuito tentar confundir os exércitos leais. Para solucionar o problema, generais devem retransmitir as mensagens recebidas várias vezes para que o efeito das mensagens de exércitos traidores seja minimizado. O algoritmo de mensagens orais é capaz de mascarar até m falhas caso existam mais do que $3m$ componentes enviando mensagens.

O problema dos generais bizantinos se baseia no estabelecimento de acordo sobre apenas um único valor estático, isto é, um valor que permanece constante durante todo o tempo de execução do algoritmo. A solução foi então adaptada para ser aplicada em algoritmos de sincronização de relógios, onde os valores envolvidos são dinâmicos. Em (LAMPORT; MELLIAR-SMITH, 1985), o autor assume que o valor inicial corresponde ao estado de um relógio lógico lido em um determinado instante de tempo. Para determinar a precisão final do sistema, o tempo de execução do algoritmo teve de ser introduzido para considerar o efeito de ρ. Executando uma vez o algoritmo de mensagens orais para transmissão do valor de relógio de cada processo envolvido faz com que, ao final, cada processo tenha em sua disposição um mesmo conjunto de valores. A partir deste conjunto de valores é feita então a escolha de um único valor de referência de acordo com algum critério de escolha, como por exemplo o valor médio ou a média dos valores.

Um grande inconveniente é que algoritmos de consistência são extremamente custosos em termos de carga de processamento nos nodos e de número de mensagens que são transmitidas através da rede. Para cada valor de relógio a ser informado, o algoritmo deve ser executado recursivamente $m + 1$ vezes onde m é o número máximo de falhas bizantinas, isto é, processos defeituosos que podem assumir um comportamento totalmente arbitrário. Considerando um sistema formado por n processos distribuídos, na primeira rodada de execução são transmitidas $n - 1$ mensagens, na segunda rodada $(n - 1).(n - 2)$, na terceira $(n - 1).(n - 2).(n - 3)$, e assim por diante.

3.4.1.2 Convergência Interativa ou Acordo Aproximado

Em algoritmos de convergência, os processos acordam sobre valores aproximados. Ao invés de retransmitir valores de relógio várias vezes, o impacto de leituras inconsistentes é amenizado pela definição de uma **faixa de aceitação**. Assumindo que existe um desvio δ_{max} permitido entre dois relógios quaisquer do sistema, o efeito de valores provenientes de nodos defeituosos é inibido descartando leituras maiores que ($C_p + \delta_{max}$), ou menores que ($C_p - \delta_{max}$), onde C_p é o valor que o relógio local de um processo p qualquer acusa no instante em que a leitura de outro relógio é feita. Se porventura uma leitura estiver fora destes limites, seu valor é automaticamente sobrescrito pelo valor de C_p naquele instante.

Sendo p e q processos não-defeituosos e C_{pr} e C_{qr} as leituras que p e q fazem de um processo r defeituoso que envia informações conflitantes para p e q, então é possível afirmar que:

$$|C_{pr} - C_{qr}| \leq 3\delta_{max}$$

O pior caso em que $|C_{pr} - C_{qr}| = 3\delta_{max}$ acontece na seguinte situação:

1. $C_p - C_q = \delta_{max}$
2. r envia para p um valor que difere $+\delta_{max}$ de C_p
3. r envia para q um valor que difere $-\delta_{max}$ de C_q

Ao final da coleta de valores de relógios, a nova referência de tempo em cada processo pode ser determinada calculando a média de todas as leituras realizadas por ele ao final de um mesmo passo de execução do algoritmo de convergência. Se existem n processos no sistema e no máximo m destes processos são defeituosos, então m leituras realizadas diferem de até $3\delta_{max}$ e as outras são iguais. Logo, a diferença máxima nas médias calculadas será limitada a $(3m/n)\delta_{max}$. Neste caso, se a condição $n > 3m$ for respeitada, então $(3m/n)\delta_{max} < \delta_{max}$, garantindo assim que o algoritmo de convergência é capaz de manter todos os relógios do sistema alinhados dentro de certos limites de precisão.

No contexto de sistemas embarcados, as propostas de sincronização de relógios mais significativas são algoritmos de convergência interativa. Isto se deve principalmente ao fato de que praticamente todas as arquiteturas de comunicação são construídas sobre um barramento serial de difusão, onde um mesmo dado transmitido é visível por todos os componentes que formam o sistema distribuído, o que restringe consideravelmente as chances de um nodo enviar uma mesma mensagem com valores conflitantes para dois ou mais outros nodos quaisquer.

3.4.2 Função de Convergência

Em (SCHNEIDER, 1986) é feita uma análise comparativa de diferentes soluções de algoritmos de sincronização propostos na literatura científica, incluindo tanto soluções de consistência quanto de convergência. O autor propõe a decomposição do cálculo da referência de tempo em duas fases, onde a primeira fase consiste na obtenção das leituras de relógios e a segunda no cálculo da referência propriamente dita a partir destas leituras. A função que recebe como entrada leituras de outros relógios e retorna uma referência de tempo recebe o nome de **função de convergência** cujo comportamento irá influenciar diretamente nas características de precisão, exatidão e tolerância à falhas da base de tempo do sistema.

As propostas de função de convergência mais significativas são apresentadas a seguir, com uma breve descrição do comportamento de cada uma e o seu impacto na precisão δ do sistema. Em todas as equações fornecidas, n é o número de leituras previamente efetuadas, ε é o erro máximo em cada leitura, ρ é a incerteza dos osciladores, R_{int} o intervalo de sincronização e m o número de leituras incorretas.

FTA: Fault-Tolerant Average

No FTA a referência de tempo é calculada pela média de todas as leituras disponíveis, excluindo àquelas referentes aos m relógios mais rápidos e os m mais lentos, onde m é número máximo de leituras incorretas dentro de um mesmo conjunto. O desvio máximo entre relógios é dado por:

$$\delta_{max} = (\varepsilon + 2\rho R_{int})\frac{n - 2m}{n - 3m}$$

Maiores detalhes podem ser encontrados em (KOPETZ; OCHSENREITER, 1987).

CNV: Interactive Convergence Algorithm

No CNV a referência de tempo é a média de todas as leituras, eliminando do cálculo aquelas cujo desvio do relógio local (em módulo) excede δ_{max}. O desvio máximo é dado por:

$$\delta_{max} = \frac{2n\varepsilon + 2\rho S(n-2m) + 2\rho n R_{int}}{n-3m}$$

onde S neste caso é o intervalo de tempo em que as leituras são feitas.

Maiores detalhes podem ser encontrados em (LAMPORT; MELLIAR-SMITH, 1985).

FTM: Fault-Tolerant Midpoint

No FTM a referência é o valor médio entre a leitura do relógio mais rápido e do mais lento, após excluir k mais rápidos e k mais lentos, onde k é um parâmetro global de configuração da função de convergência. O desvio máximo é dado por:

$$\delta_{max} = 4(\varepsilon + 4\rho R_{int})$$

Maiores detalhes podem ser encontrados em (WELCH; LYNCH, 1988).

DCA: Daisy-Chain Algorithm

No DCA a cada nova leitura obtida sua utilização é imediata, podendo ser descartado logo em seguida. A função de convergência recebe como entrada o valor de δ_{ij}, isto é, o desvio entre a última leitura obtida e o relógio local. Se o desvio entre a leitura e o relógio local excede δ_{max} então a leitura é descartada senão a referência é a própria leitura. Neste caso o desvio máximo é dado por:

$$\delta_{max} = 2\varepsilon + \rho R_{int}$$

Maiores detalhes podem ser encontrados em (LONN; SNEDSBØL, 1995) e (LONN, 1999b).

A formalização destes métodos de sincronização costuma assumir que todas as leituras de relógios distribuídos, cálculo do fator de correção e aplicação deste no relógio local são ações atômicas, executadas em um mesmo instante de tempo. Contudo, em plataformas de comunicação cuja meio físico consiste em um barramento serial, todas as mensagens são transmitidas e processadas em seqüencia ao longo do intervalo de sincronização R_{int}. Além disso, o algoritmo de sincronização leva um certo tempo para ser executado de forma que no momento em que a função de convergência é chamada os valores de relógios já estão desatualizados.

Ao invés de trabalhar diretamente com valores de relógios, a melhor alternativa é trabalhar com os desvios relativos referentes às leituras realizadas para que estas informações tenham validade durante um período de tempo maior (LAMPORT; MELLIAR-SMITH, 1985). A cada leitura calcula-se a diferença existente entre o relógio local e o relógio de outro processo. Conseqüentemente, a função de convergência quando aplicada a um conjunto de desvios retorna um **fator de correção** a ser aplicado na base de tempo local, ou seja, o relógio será adiantado ou subtraído.

3.4.3 Intervalo de Sincronização

Assumindo que ρ_{max} seja fixado a um valor determinado pelo oscilador de menor custo utilizado no sistema, a divergência entre os relógios aumenta ao longo do tempo na seguinte proporção:

$$\simeq 2\rho_{max}\Delta t$$

Supondo que, imediatamente após uma correção o desvio máximo do sistema seja de $\delta = \delta_{init}$, então, o desvio imediatamente antes da próxima correção é dado por:

$$\delta_{max} = \delta_{init} + 2\rho_{max}R_{int} \tag{3.6}$$

onde R_{int} é o espaço de tempo entre duas correções consecutivas chamado de **intervalo de sincronização** e o valor de δ_{init} uma constante depende da função de convergência, i.e. do algoritmo de sincronização utilizado. A partir desta relação fica claro que, quanto menor for o espaço entre as correções, menor será o efeito de ρ_{max} sobre δ_{max}. Mesmo que ρ_{max} seja relativamente pequeno, por exemplo na ordem de 10^{-6}, na medida que R_{int} aumenta o termo $2\rho_{max}R_{int}$ começa a impactar negativamente sobre a precisão do sistema.

3.4.4 Erro na Leitura de Relógios

O tempo que uma mensagem leva para ser transmitida de um processo a outro é chamado de **atraso de propagação** e consiste na principal fonte de erro na leitura de relógios distribuídos. Imaginemos que em um certo instante de tempo T o processo p decide enviar uma mensagem com o valor do seu relógio local para um outro processo q qualquer conforme mostra a Figura 3.4. Caso q assuma diretamente o valor T como sendo o valor do relógio de p (C_p) no momento em que a mensagem é recebida, irá existir um erro nesta leitura que é função de basicamente três fatores:

1. Atraso de saída Δ_{TX} do nodo transmissor, definido como sendo o atraso entre a solicitação da transmissão e o instante em que o primeiro bit do quadro é colocado no barramento.

2. Atraso da rede Δ_{bus}, referente ao tempo de transmissão do quadro no meio físico.

3. Atraso de entrada Δ_{RX} do nodo receptor, definido como sendo o tempo que a pilha de rede leva para entregar a mensagem recebida para o host.

As componentes Δ_{RX} e Δ_{TX} dependem fundamentalmente dos atrasos introduzidos e da velocidade de processamento na pilha do protocolo em questão, enquanto que o atraso da rede depende do tamanho dos quadros, da taxa de transmissão no meio físico e da topologia utilizada. Em redes altamente desconectadas, a componente Δ_{bus} sofre maiores variações em razão dos diferentes atrasos causados pela passagem das mensagens por roteadores, pontes ou hubs intermediários. Quando a topologia é do tipo barramento simples, a variação de Δ_{bus} é pequena, resultado da distância relativa entre dois nodos. Na topologia em estrela, a característica é a mesma porém, deve ser incluído ainda o atraso intrínseco do hub central.

O atraso de propagação não pode ser eliminado mas ele pode ser **compensado**, somando uma estimativa deste atraso ao valor da leitura. Em protocolos de tempo-real onde os atrasos de descida e subida na pilha podem ser previamente estabelecidos, é possível

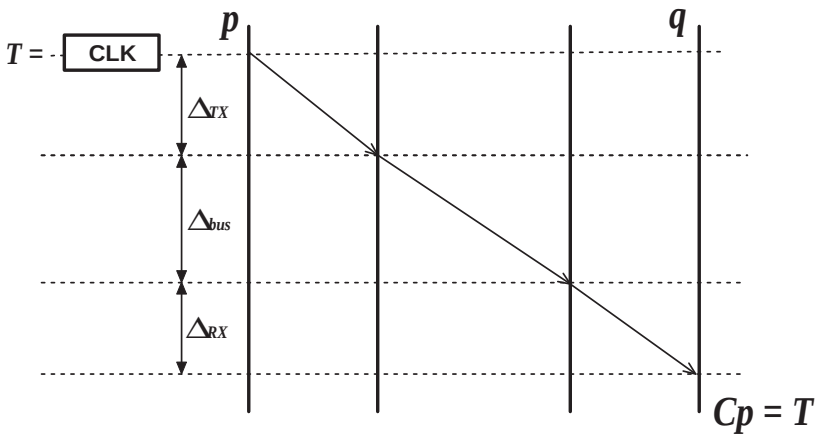

Figura 3.4: Componentes do atraso de propagação.

determinar limites mínimo e máximo para o atraso de propagação de um dado sistema, onde:

$$\Delta_{pq.max} = \Delta_{TX.max} + \Delta_{bus.max} + \Delta_{RX.max}$$

$$\Delta_{pq.min} = \Delta_{TX.min} + \Delta_{bus.min} + \Delta_{RX.min}$$

Na prática é possível fazer com que:

$$\Delta_{TX.max} = \Delta_{TX.min}$$
$$\Delta_{RX.max} = \Delta_{RX.min}$$

sobretudo em plataformas de comunicação embarcadas ou industriais onde somente as camadas 1, 2 e 7 são implementadas sobre uma topologia de rede do tipo barramento ou estrela. Em vista disso, a diferença entre $\Delta_{pq.max}$ e $\Delta_{pq.min}$ fica por conta de Δ_{bus} que varia em função da distância relativa entre os nodos do sistema, supondo que todos os quadros são de mesmo tamanho. Para que as leituras sejam mais precisas seria então necessário configurar um fator de compensação distinto para cada dois pares de nodos p e q. Contudo, uma alternativa mais simples é utilizar o valor médio e fazer deste valor um fator de compensação global, ou seja:

$$\kappa = \frac{\Delta_{pq.max} + \Delta_{pq.min}}{2}$$

Assim, a cada leitura o erro que estará sendo feito fica limitado a:

$$\varepsilon = \frac{\Delta_{pq.max} - \Delta_{pq.min}}{2} \simeq \frac{\Delta_{bus.max} - \Delta_{bus.min}}{2}$$

Supondo C_{qp} a leitura que um processador q faz do relógio de p, então:

$$C_{qp} = T + \kappa \pm \varepsilon$$

onde T é o valor de relógio enviado por p no corpo de sua mensagem e κ o **fator de compensação da leitura**. Assim, garante-se que o leitor tenha uma idéia mais apurada do estado da base de tempo do transmissor.

3.4.5 Correção de Relógios Locais

Existem basicamente duas maneiras de aplicar o fator de correção em um relógio lógico. Quando a correção se caracteriza por uma mudança súbita no valor do relógio em um determinado instante de tempo t_c, então a correção é chamada de **correção discreta**, ou correção por **ajuste de fase**.

Uma maneira simples e intuitiva de compreender o comportamento de relógios que sofrem correções de fase é considerar que, ao final de cada intervalo de sincronização i, um novo relógio C^{i+1} é criado a partir do valor atual do relógio C^i somando o fator de correção Λ^i recém calculado pelo algoritmo, ou seja:

$$C^{i+1}(t_c) = C^i(t_c) + \Lambda^i$$

onde t_c é o instante em que a correção acontece. Assim, a visão do tempo do sistema distribuído fica "recortada", isto é, o tempo lógico é particionado em intervalos de mesma duração dentro dos quais os relógios locais são atualizados livremente por dispositivos osciladores sofrendo a influência de ρ.

Observa-se que na correção por ajuste de fase a base de tempo sofre uma modificação instantânea onde Λ unidades de tempo lógico ou desaparecem, quando o relógio for adiantado, ou acontecem duas vezes no sistema, no caso do relógio precisar ser defasado fazendo com que C deixe de ser uma função contínua e monotônica. Como conseqüência, o sistema pode apresentar um comportamento inconsistente caso algum evento tenha sido programado para algum instante de tempo próximo do momento em que os relógios são corrigidos. Se este evento representasse, por exemplo, a ativação de uma tarefa, então um salto a frente no tempo lógico poderia fazê-lo desaparecer, enquanto que um salto para trás faria com que esta tarefa fosse indevidamente ativada mais de uma vez. Em (LAMPORT; MELLIAR-SMITH, 1985) é sugerido que para valores de Λ muito menores do que o intervalo de sincronização R_{int}, a solução mais simples é de garantir em tempo de projeto que nada seja programado para acontecer no sistema durante um pequeno intervalo em torno do instante em que o fator de correção é aplicado.

No intuito de contornar o problema de uma maneira mais "elegante", algumas propostas, como é o caso de (SCHMUCK; CRISTIAN, 1990), defendem o uso de uma técnica denominada de **amortização**, ou correção por **ajuste de período**. Ao invés de sofrer uma alteração súbita, o relógio é acelerado ou retardado por um fator α durante um intervalo de tempo que varia em função do tamanho do fator de correção calculado. Para isto, deve existir obrigatoriamente um contador de ciclos de máquina intermediário entre a fonte primária e o contador do relógio local para que a taxa de atualização da base de tempo possa ser modificada.

A "suavidade" do processo de amortização é função direta do valor de α. Quanto menor for α, mais suave é a correção, ou seja, a variação de velocidade que um relógio sofre é mais sutil. No entanto, em (SCHMUCK; CRISTIAN, 1990) é mostrado que a precisão do sistema é prejudicada para valores pequenos de α. A Figura 3.5 apresenta graficamente o comportamento de um relógio discreto em comparação com um relógio amortizado. Considerando que $R = t_1 - t_0$ representa o intervalo de tempo entre duas correções consecutivas, o relógio amortizado deve "alcançar" o equivalente discreto antes do instante t_1, caso contrário um erro acumulativo pode fazer com que a diferença máxima entre dois relógios quaisquer exceda δ_{max}.

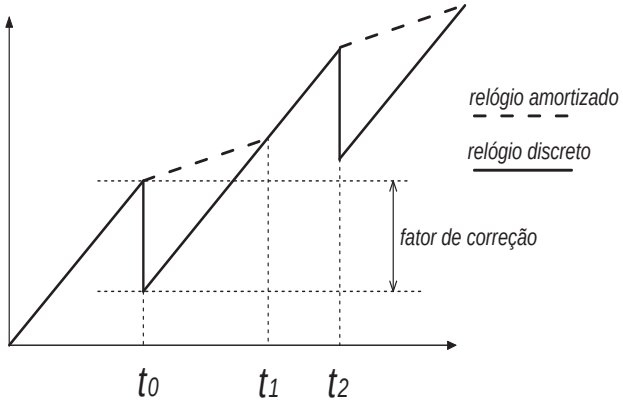

Figura 3.5: Comparação entre amortização e correção discreta.

3.5 Sincronização em Protocolos TDMA

Em (RAMANATHAN; SHIN; BUTLER, 1990) o autor destaca que em muitos casos o que se busca na prática é um meio termo entre a precisão de soluções em hardware e o baixo custo associado a soluções em software, motivando assim a implementação de sistemas híbridos onde o algoritmo de sincronização é implementado na forma de uma combinação de hardware e de software. Tais sistemas são caracterizados pela ausência de interconexões dedicadas para sinais de relógios mas que no entanto são construídos com suporte em hardware.

Protocolos com controle de acesso TDMA são exemplos práticos onde o algoritmo de sincronização é implementado em uma forma híbrida. O mecanismo de leitura, a função de convergência e o ajuste do relógio local estão fortemente acoplados com circuitos específicos que operam em nível físico e de enlace.

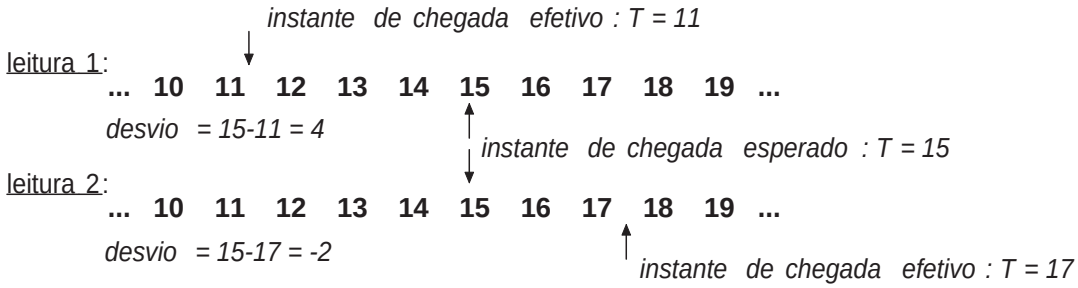

Figura 3.6: Princípio de leitura indireta.

Uma característica favorável em protocolos TDMA é que todas as transmissões são previamente agendadas sobre uma linha de tempo comum, assumindo que durante todo o tempo de vida do sistema os nodos estão com seus relógios sincronizados dentro de certos limites de precisão. Logo, o princípio de **leitura indireta** ilustrado na Figura 3.6 pode ser utilizado na obtenção dos desvios relativos entre os nodos. A diferença entre o instante de chegada esperado e o instante de chegada efetivo é proporcional ao desvio entre os relógios naquele exato momento. Se o desvio calculado pelo receptor for negativo então a conclusão é que ele está adiantado em relação a base de tempo do transmissor. Da mesma forma, se o desvio for positivo então significa que o receptor está atrasado.

A informação de desvio é temporariamente armazenada para ser utilizada em seguida

no cálculo da função de convergência de forma que não existe de fato a necessidade de transmitir valores de relógio explícitos no corpo das mensagens. A maior vantagem do método de leitura indireta é que não existe carga adicional da rede.

Em protocolos TDMA é possível atingir sincronização interna com granularidade de 1μs ou até mesmo menor. Utilizando a plataforma CASCA consegue-se criar e manter uma base de tempo global com precisão na ordem de 0.5μs. Para isso, o uso de hardware dedicado é indispensável na obtenção das leituras em virtude da necessidade de se detectar com alta precisão o instante em que a transmissão é iniciada e assim inferior o desvio associado com menor erro possível.

3.6 Inicialização da Base de Tempo Global

Mesmo que algoritmos de sincronização considerem que inicialmente os relógios de todos os processos envolvidos estão suficientemente próximos, sabe-se que na prática a realidade é outra. Em algum momento os componentes que formam o sistema foram energizados pela primeira vez, alguns deles podendo ter sido ligados ao mesmo tempo, uns pouco antes, outros pouco depois. Ainda que todos os relógios tenham sido iniciados com o mesmo valor, sabe-se que é extremamente complicado, ou até mesmo impossível, fazer com que todos os nodos entrem em operação ao mesmo tempo em um ambiente essencialmente distribuído.

Imediatamente após o sistema ter sido energizado, não existe nenhuma garantia de que os desvios relativos entre relógios sejam menores que δ_{max}, condição primária para o funcionamento de qualquer algoritmo de sincronização. Logo, o sistema deve também possuir um mecanismo de criação inicial de uma base de tempo que traga os relógios locais de todos os nodos envolvidos para dentro do limite de precisão previamente estabelecido.

Em protocolos TDMA, o desalinhamento inicial de relógios faz com que colisões possam ocorrer. Se a base de tempo é centralizada, a inicialização é trivial bastando colocar o nodo mestre em operação para que os demais nodos se integrem a ele, tal como no TTCAN que será apresentado no capítulo 4. Os demais nodos escravos possuem um comportamento passivo na ausência do mestre e só entram em ação se o mestre transmite uma mensagem de referência marcando o passo do esquema TDMA.

A inicialização de sistemas com base de tempo descentralizada é mais complexa pois existe o perigo de que colisões aconteçam após dois ou mais nodos terem sido ligados, o que pode atrasar ou até mesmo impedir que a sincronização inicial seja obtida. Quando uma colisão ocorre, algum procedimento deve ser tomado de forma a evitar a próxima colisão. Estritamente falando, a inicialização da base de tempo deve ser determinística, ou seja, o intervalo entre o momento que um nodo é ligado e o instante em que ele entra em regime normal de operação deve ser limitado. Caso este mecanismo não esteja preparado para lidar com possíveis colisões, a criação da base de tempo pode atrasar indefinidamente. O processo de inicialização também pode ser chamado após os nodos terem sido energizados pela primeira vez em caso de interferência excessiva no meio físico. Caso a comunicação seja comprometida por um intervalo de tempo significativo sem que nenhuma correção seja feita, o desvio máximo entre dois relógios pode ultrapassar os limites de δ_{max}.

A solução para o problema da criação da base de tempo está na atitude tomada pelo protocolo quando alguma colisão inicial é detectada. Em linhas gerais, basta que ao menos um nodo consiga transmitir uma mensagem livre de colisões para que os demais

consigam adotar a base de tempo do primeiro transmissor bem sucedido.

Em (LONN, 1999a) são apresentados três métodos para a solução deste problema: *Zero First*, *Inc/Dec* e *Lock-step*. O procedimento utilizado nesses três métodos e bastante semelhante. Existe um período de silêncio durante o qual cada nodo aguarda por alguma atividade no canal de comunicação. Após esta espera inicial, o nodo faz uma tentativa de transmissão, no intuito de transmitir a primeira mensagem sem colisões. A diferença está no cálculo do instante da próxima tentativa caso alguma colisão seja detectada.

4 ESTADO DA ARTE & TRABALHOS RELACIONADOS

4.1 A Arquitetura Time-Triggered

A arquitetura time-triggered, ou somente TTA, engloba um modelo completo de projeto e funcionamento de sistemas distribuídos com características de tempo-real. A proposta mais atual apresentada em (KOPETZ; BAUER, 2003) é resultado de um extenso trabalho desenvolvido no contexto do projeto MARS, iniciado em 1979 na Escola Técnica de Berlin, sob a liderança do Prof. Dr. Hermann Kopetz. Em 1995 um trabalho de cooperação com a DaimlerChrysler resultou em um estudo de caso onde a TTA foi empregada no protótipo de um veículo equipado com sistema por-fio embarcado. Contudo, sua utilização tem sido também de grande interesse em outras áreas. Em 1999, a Alcatel deu início a um trabalho interno de pesquisa visando a implementação de sistemas de controle para trens. Além disso, em 2000, a arquitetura TTA foi adotada pela Honeywell para desenvolvimento de sistemas de navegação em aeronaves, abrindo caminho para o uso de componentes COTS (commercial of-the-shelf), i.e. componentes de propósito geral, em projeto de sistemas altamente críticos para uso neste domínio.

Desde sua concepção inicial até os dias de hoje, o modelo de comunicação da TTA, i.e. modelo time-triggered (TT), tem sido adotado como princípio básico no projeto de diversos protocolos destinados ao projeto de sistemas de controle distribuídos.

4.1.1 Modelo de Comunicação Time-Triggered

O princípio fundamental do modelo de comunicação time-triggered estabelece que as ações mais relevantes do sistema são disparadas em instantes previamente definidos na medida em que o tempo avança, conforme ilustra a Figura 4.1. Em se tratando de sistemas distribuídos, onde as unidades de processamento estão fisicamente separadas no espaço,

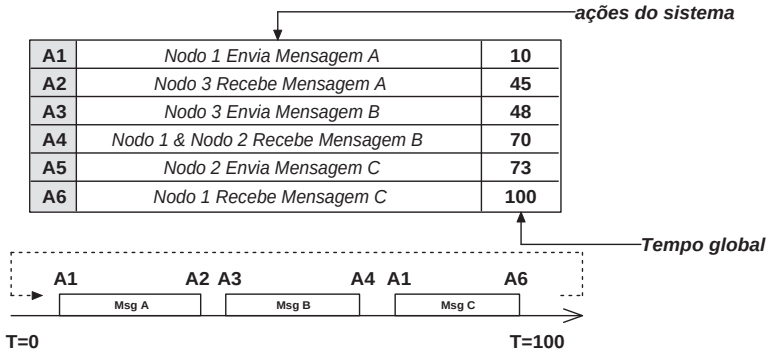

Figura 4.1: Modelo de comunicação time-triggered.

fica claro a necessidade de uma base de tempo global sincronizada. Em razão disso, cada controlador de rede da TTA possui um mecanismo de sincronização de relógios implementado com uso de hardware dedicado. De acordo com a versão do protocolo utilizado, o mecanismo de sincronização pode apresentar uma característica centralizada como na TTP/A, versão mais simples de baixo custo, ou essencialmente distribuído, como no caso da versão TTP/B e da versão TTP/C tolerante à falhas, a qual será apresentada em maiores detalhes nesta seção.

O tempo de acesso ao barramento é particionado em slots dedicados segundo um mecanismo TDMA. Todos os slots têm o **mesmo tamanho** que precisa ser definido na fase de projeto em função do tempo de transmissão do maior quadro e da velocidade do controlador de rede na execução das funções primitivas da plataforma.

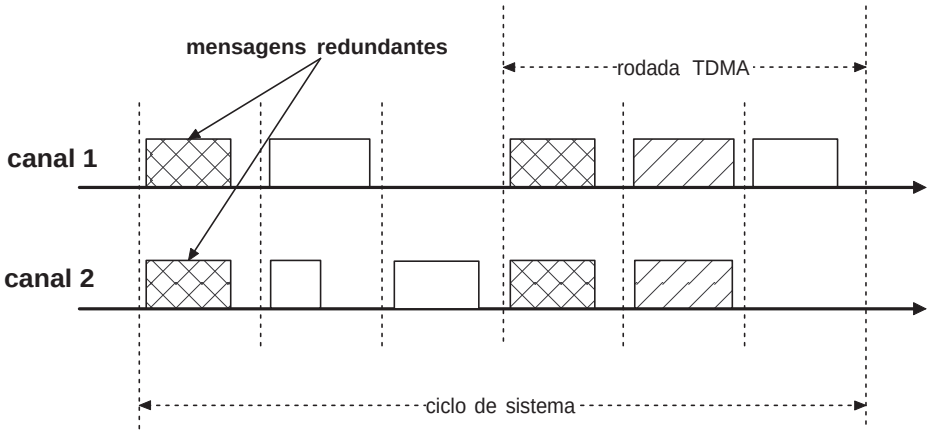

Figura 4.2: Ciclo de cluster – comunicação em barramentos replicados.

A seqüencia na qual cada nodo possui um slot de tempo exclusivo para transmitir é chamada de **rodada** TDMA. Após o término de uma rodada uma outra com uma seqüencia diferente de mensagens é iniciada [1]. O conjunto de todas as rodadas previstas recebe o nome de **ciclo de sistema**, ou **ciclo de cluster**, que consiste num horizonte de tempo finito dentro do qual as ações do sistema são agendadas. Quando o ciclo de sistema chega ao seu fim, a contagem do tempo volta novamente ao seu início. A Figura 4.2 mostra um ciclo de sistema onde mensagens redundantes são transmitidas em barramentos replicados.

Sistemas construídos com base no modelo TT são aqueles onde predominam requisitos de comunicação síncrona e regular, com atividades sistemáticas de consumo e produção de dados que são válidos por um curto período de tempo. Estes dados provém de grandezas físicas do objeto ou processo controlado denominadas de entidades temporeal, ou somente **entidades RT**. Quando amostradas as entidades RT são convertidas no formato digital para que o valor do seu estado possa ser interpretado pelo sistema. As estruturas de dados que comportam amostras de entidades RT por sua vez recebem o nome de **variáveis de estado** ou **imagens**.

A maior crítica feita sobre sistemas TT é quanto a sua flexibilidade. Um sistema onde os instantes de transmissão têm de ser previamente definidos ao longo do tempo não é facilmente expansível, uma vez que para qualquer eventual inclusão de um nodo que não tenha sido prevista durante a fase de projeto todo o sistema deve ser reprogramado. No entanto, a definição estática do comportamento da comunicação sob uma perspectiva global oferece as seguintes vantagens:

[1] Exceto a configuração elementar onde só existe uma rodada. Neste caso, a mesma rodada se repete.

1. Detecção de erros simples e rápida.

   Todas as unidades de processamento ficam constantemente observando o comportamento do sistema e verificando se este comportamento coincide com o que foi previsto durante a fase de projeto. O não recebimento de uma mensagem no instante esperado pode ser imediatamente interpretado como um erro do transmissor, por exemplo.

2. Latências fixas.

   Fazendo a camada de aplicação "acompanhar" o comportamento da plataforma de comunicação é possível reduzir ao mínimo a variabilidade da latência de rede. A tarefa responsável pelo cálculo de um resultado ou obtenção de uma amostra digital deve ser agendada para execução imediatamente antes do instante de transmissão de sua mensagem.

4.1.2 Modelo de Comunicação Event-Triggered

A partir da definição da TTA os protocolos de característica assíncrona, como por exemplo CAN e Ethernet, passaram a ser classificados protocolos **event-triggered** no intuito de facilitar o estudo e o projeto de sistemas distribuídos de tempo-real.

Sistemas ET não garantem particionamento temporal visto que o tempo de resposta de cada subsistema é função do comportamento da carga na rede ao longo do tempo (DILGER et al., 2005). Ao contrário de sistemas TT, o acesso ao meio não se dá de forma exclusiva sendo que qualquer nodo pode tentar transmitir uma mensagem a qualquer instante com o risco de contenção. Apesar de apresentarem variações de tempo maiores, sistemas ET apresentam uma reação mais rápida a eventos assíncronos, sobretudo quando a carga da rede é baixa.

Na TTA, existe a possibilidade de desenvolver suporte em alto nível para transmissão assíncrona de dados, ou seja, eventos de natureza não determinística como por exemplo tarefas de configuração, diagnóstico, manutenção e alarmes. Filas de entrada e saída são gerenciadas em software e os dados são transmitidos utilizando slots de tempo vazios que podem resultar da definição do agendamento global da comunicação.

4.1.3 Metodologia de Projeto

A TTA propõe um desacoplamento quase total entre a camada da aplicação e o protocolo visando evitar ao máximo que falhas de projeto ou falhas físicas provenientes de componentes COTS interfiram na atividade de comunicação do sistema. Em regime normal de operação, o acesso aos sinais de controle da plataforma é altamente restrito. A plataforma de comunicação funciona de forma autônoma, sem a necessidade de comandos de transmissão provenientes das tarefas. Funções de sincronização de relógios, controle de acesso ao meio e todas as demais primitivas ficam confinadas nos controladores de rede e são executadas de maneira transparente para o host.

O projeto de sistemas distribuídos é dividido em duas fases: configuração da plataforma de comunicação e desenvolvimento do código da aplicação. Sabendo que o comportamento da comunicação é determinado antes de colocar o sistema em operação, na primeira fase é definido um escalonamento **estático** da utilização do meio físico. Devem ser configurados todos os instantes de início e término dos slots de tempo assim como os endereços da memória de onde o conteúdo das mensagens é lido no lado transmissor e gravado no lado dos receptores. Estes endereços se referem a posições da memória de interface com o host denominada de CNI (Communication Network Interface).

Toda a informação de configuração resultante da primeira faze é inserida em uma estrutura de dados interna dos controladores de redes chamada de MEDL (Message Descriptor List) cujo tamanho é diretamente proporcional ao tamanho do ciclo de sistema[2].

A segunda fase do projeto consiste no desenvolvimento **modularizado** do software da camada da aplicação, guiado pela definição do escalonamento estático resultante da primeira fase. Subsistemas podem ser implementados separadamente uma vez que a TTA garante particionamento temporal conforme já discutido no capítulo 2. Em regime normal de operação o acesso às imagens RT é feito como se fossem variáveis locais disponíveis em endereços de memória da CNI sem a necessidade do uso de chamadas do tipo *send()* e *receive()*.

4.1.4 Protocolo TTP/C

Na literatura as siglas TTA e TTP muitas vezes são usadas de maneira indiscriminada. Todavia, o protocolo TTP consiste na plataforma de comunicação da TTA. Em decorrência dos diferentes requisitos de segurança das aplicações, três versões foram propostas: o TTP/C, versão tolerante à falhas para uso em aplicações críticas e duas versões de baixo custo, a TTP/A e a TTP/B, para aplicações que não estão diretamente envolvidas com a integridade de seres humanos ou de propriedade de alto valor agregado. Mesmo que as três versões apresentem discrepâncias relacionadas ao suporte para tolerância à falhas, o paradigma de comunicação adotado é o mesmo, ou seja, nos três casos utiliza-se uma base de tempo global para execução de um escalonamento estático da comunicação conforme um mecanismo TDMA. A versão TTP/C no entanto é a mais completa, com serviços especiais para detecção de erros e replicação de componentes, além de uma base de tempo global essencialmente distribuída. As demais versões do protocolo TTP implementam apenas um subconjunto simplificado de serviços.

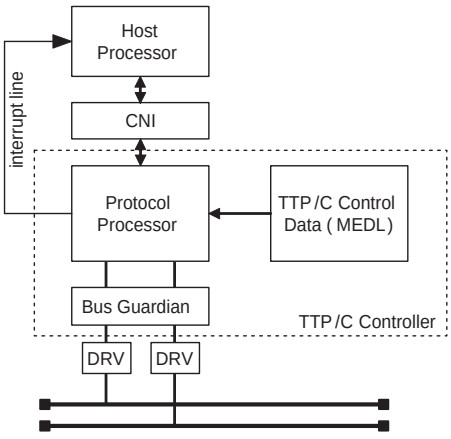

Figura 4.3: Organização interna do controlador TTP/C.

A estrutura de interconexão do TTP/C é replicada (Figura 4.3), formada por dois canais de difusão serial de dados em uma topologia do tipo barramento serial, estrela ou uma combinação de ambos. A CNI é implementada na forma de memórias dupla porta enquanto a MEDL é uma memória somente de leitura. Sistemas TT são extremamente sensíveis à falhas no domínio do tempo logo, existe um dispositivo de guarda (bus guardian) sintetizado no mesmo encapsulamento do controlador de rede TTP/C para proteger os

[2]O limite de memória disponível para esta estrutura de dados é um fator que restringe a capacidade de configuração e expansão do sistema.

canais seriais contra nodos defeituosos que porventura possam tentar monopolizá-los. Na Figura também é mostrado que pelo menos um sinal de interrupção deve ser levado diretamente ao host para que este possa acompanhar o comportamento global da comunicação do sistema.

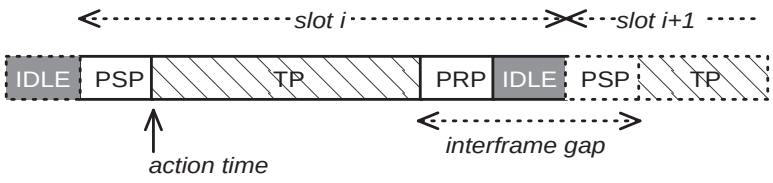

Figura 4.4: Fases de processamento em um slot da TTA.

Toda e qualquer transmissão tem início nos chamados instantes de ação (*action times*) que delimitam as fronteiras de tempo entre os slots. Na Figura 4.4 a fase de transmissão (TP) representa o tempo necessário para que um quadro seja transmitido. Antes de dar início à transmissão propriamente dita existe uma fase de pré-transmissão (PSP) durante a qual a MEDL é acessada para consulta de informações referentes ao respectivo slot de tempo. Quando a transmissão é finalizada, o controlador entra na fase de pós-recepção (PRP) na qual as funções primitivas da plataforma são executadas tais como cálculo de CRC e sincronização de relógios.

O tamanho de um slot é definido como a soma de TP com o espaço mínimo entre quadros (*interfame gap*), que consiste em uma estimativa do tempo máximo que um controlador necessita para execução da PSP e PRP. O tempo não utilizado por TP, PSP e PRP é definido como IDLE, durante o qual o controlador TTP/C fica ocioso. O tempo de IDLE sempre irá existir pois a duração exata de PSP e PRP em cada nodo é imprevisível e os valores utilizados na definição do tamanho do slot expressam o pior caso. Para evitar conflitos de acesso na CNI entre o controlador e o host o último deve acompanhar a base de tempo e a CNI é modificada somente durante as fases TP e IDLE.

4.1.5 Criação da Base de Tempo

A primeira atitude de um nodo ao ser energizado pela primeira vez é a de observar a atividade nos canais de comunicação buscando adaptação a uma base de tempo já existente. Caso já exista comunicação em curso o nodo integra-se a mesma visão de tempo dos demais. Para isto alguns quadros carregam explicitamente informação sobre o valor da base de tempo do seu nodo transmissor respectivo.

Um nodo em processo de integração a uma base já existente aguarda durante um certo período de tempo dentro do qual ele espera receber um quadro que carrega explicitamente uma leitura de relógio. A especificação do protocolo TTP/C determina que pelo menos um quadro com informação explícita de tempo deve ser transmitido a cada duas rodadas TDMA. Caso nenhum quadro seja recebido, o nodo inicia o procedimento de criação de uma nova base de tempo denominado de **entrada a frio**.

Durante o procedimento de entrada a frio garante-se que as colisões não se propagam pela atribuição de um tempo de espera distinto para cada nodo do cluster. Em caso de colisão, uma vez detectada [3], todos os nodos reiniciam o tempo de espera ao mesmo tempo. A partir de então, garante-se que novas colisões não ocorrem uma vez que cada nodo possui um tempo de espera distinto, configurado em tempo de projeto na MEDL.

[3] A implementação da camada física deve ser provida da capacidade de detectar colisões.

4.1.6 Sincronização de Relógios

A unidade mínima de representação do tempo no TTP/C é o microtick, extraído diretamente da fonte primária de tempo do nodo. A duração de 1 microtick não é necessariamente a mesma para todo o sistema visto que nodos podem ter dispositivos osciladores com diferentes freqüências. No entanto, o protocolo necessita de uma unidade homogênea para especificação do seu comportamento ao longo do tempo: o macrotick. O tempo de duração nominal desta unidade é um parâmetro global que deve ser configurado em cada controlador com o número de microticks necessários. Todas as relações de tempo na camada da aplicação tais como períodos, deadlines e instantes de ativação devem ser expressas em unidades de macroticks.

O protocolo TTP/C utiliza o algoritmo FTA (KOPETZ; OCHSENREITER, 1987) para cálculo do fator de correção a partir de 4 leituras indiretas recolhidas ao longo de um intervalo de sincronização R_{int}. Nem todas as leituras são utilizadas. Na fase de projeto são selecionados os nodos que possuem osciladores mais precisos para servirem de referência de tempo logo, somente as leituras referentes a estes nodos contribuem no cálculo do fator de correção Λ. De acordo com a especificação do protocolo (TTP/C, 2003), deve haver um número mínimo de quatro nodos de referência para que o algoritmo de sincronização seja tolerante à falhas. O algoritmo identifica quem são estes quadros consultando o estado de uma flag de controle (SYF Flag) do respectivo slot de tempo disponível na MEDL. Caso esta flag esteja ativada, o controlador armazena a leitura para cálculo do fator de correção caso contrário a leitura é descartada.

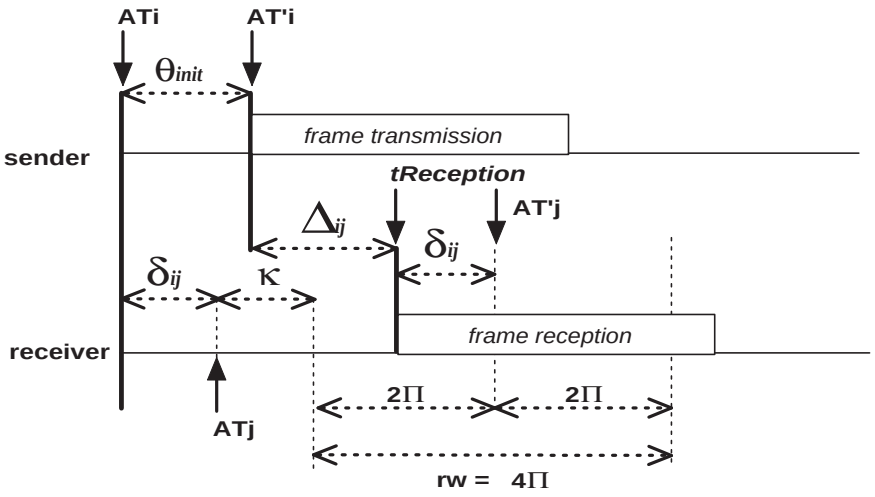

Figura 4.5: Mecanismo de leitura indireta no TTP/C.

O procedimento de leitura indireta no TTP/C é mostrado na Figura 4.5. No lado transmissor, ATi é o instante de transmissão e AT'i é o instante de transmissão deslocado a partir do qual o quadro começa efetivamente a ser transmitido. Esta espera inicial serve para evitar que um receptor mais lento receba o início de quadro antes mesmo de estar apto a realizar a leitura. No lado receptor, ATj é o instante de transmissão do quadro (do ponto de vista do seu relógio local) e AT'j é o instante no qual a detecção do início de quadro é esperada. É definida uma janela de recepção (**rw**) cujo centro coincide com AT'j. Somente são consideradas leituras que não ultrapassem os limites desta janela de recepção, i.e. o desvio δ_{ij} entre os relógios não pode ser maior que 2Π nem menor que -2Π, onde Π representa a precisão da base de tempo global, i.e. $\Pi = \delta_{max}$. A distância de

tempo entre AT'j e ATj é equivalente a soma do deslocamento inicial no transmissor com o atraso de propagação na rede, isto é:

$$\text{AT'j - ATj} = \kappa + 2\Pi = \Theta_{init} + \Delta_{ij}$$

onde κ é a constante de compensação da leitura.

O FTA é executado durante a fase PRP (Figura 4.4) nos slots configurados com a "ClkSyn Flag" na MEDL ativada. As leituras válidas são armazenadas em uma pilha de profundidade 4. Quando o FTA é aplicado, o menor e o maior desvio são descartados, é feita a média dos 2 valores restantes e como resultado têm-se o fator de correção Λ a ser aplicado na base de tempo local. Caso o módulo do fator de correção calculado seja maior que $\frac{\Pi}{2}$ o controlador envia um sinal para o host e cancela a transmissão de suas mensagens até que um novo fator de correção adequado seja obtido.

O fator de correção é aplicado por amortização. A velocidade do relógio local é alterada por um curto espaço de tempo fazendo o tamanho do macrotick aumentar ou diminuir. Para isto deve ser definida a suavidade α que representa o número máximo de microticks a serem inseridos ou retirados de cada macrotick, e o número $frMT$ de macroticks consecutivos que são modificados.

De acordo com a especificação do protocolo, estes parâmetros estão relacionados ao intervalo de sincronização da seguinte forma:

$$R_{int} \geq \alpha((frMT + 1)\Pi + \mu T)$$

onde μT é o tamanho de 1 microtick e Π a precisão do sistema igual ao tamanho de 1 macrotick.

Para o FTA a seguinte relação também é valida:

$$\Pi = 2\rho R_{int} + \Phi$$

na qual Φ está ligado ao efeito da função de convergência do algoritmo FTA, que pode ser determinado a partir de:

$$\Phi = \frac{\Pi + 2\varepsilon}{2}$$

onde ε é a diferença entre o atraso mínimo e máximo (erro na leitura), logo:

$$\Pi = 4\rho R_{int} + 2\varepsilon$$

Observa-se que a equação de Π é idêntica a equação de δ_{max} do FTA mostrada no capítulo 3 para $n = 4$ e $m = 1$, ou seja, no pior caso somente uma das quatro leituras armazenadas na pilha é inconsistente.

4.2 FlexRay

O protocolo FlexRay (FLEXRAY, 2004) é resultado do trabalho conjunto do grupo FlexRay Consortium formado pela BMW, DaimlerChrysler, Motorola, Philips, General Motors e Bosch. Por trás desta iniciativa existe de fato um grande interesse econômico visando competir com a TTA no mercado. Até o surgimento do FlexRay o protocolo TTP/C representava a solução mais adequada, senão a única, para o projeto integrado de sistemas críticos em sistemas embarcados distribuídos de grande porte visando a comercialização.

A proposta busca oferecer uma solução com maior flexibilidade no projeto, configuração e expansão de sistemas distribuídos ao mesmo tempo que o particionamento temporal é garantido por um esquema de comunicação parcialmente time-triggered. Ao contrário da TTA, no FlexRay existe a possibilidade de que os nodos compartilhem parte da capacidade de transmissão do canal de comunicação. Devido a esta característica, um sistema FlexRay é mais fácil de ser modificado em tempo de execução, ou seja, novos nodos com novas funcionalidades podem ser conectados ao canal de comunicação sem a necessidade de se reprogramar todos os componentes do sistema, mais importante, sem interferir na parte time-triggered.

No que se refere a infraestrutura física de rede, um sistema FlexRay pode ser construído sobre uma topologia do tipo barramento, estrela (fibra ótica), ou uma combinação de ambos. A duplicação do canal de comunicação é **opcional** e depende do nível de confiabilidade exigido pela aplicação.

A especificação atual determina que a taxa de transmissão vai de 2.5Mbps até 10Mbps. Esta limitação está diretamente ligada a dificuldade de programação dos parâmetros de comunicação que um sistema com controle de acesso TDMA impõe, principalmente naquilo que se refere ao mecanismo de sincronização de relógios. A documentação do protocolo fornece valores padrão para alguns destes parâmetros. Para configurações com velocidade fora da faixa especificada o correto funcionamento não é garantido pelo fabricante.

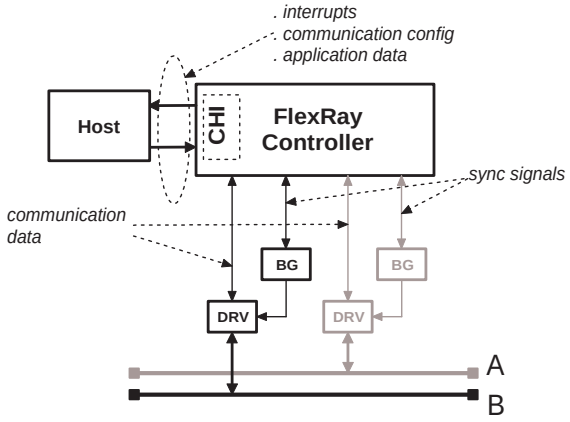

Figura 4.6: Esquema simplificado de um nodo FlexRay.

Não são fornecidos maiores detalhes quanto a organização interna do controlador. Contudo, algumas conclusões podem ser inferidas a partir do esquema simplificado de um nodo FlexRay mostrado na Figura 4.6. Por exemplo, dispositivos de guarda são implementados em encapsulamentos distintos, um para cada canal, posicionados fora do caminho serial do sinal de dados. Sua função é de habilitar/desabilitar os transdutores (DRVs) nos instantes certos. Os componentes em tonalidade mais fraca indicam recursos opcionais para replicação do canal de comunicação.

A interface de comunicação com o processador principal (host) é uma memória interna ao controlador denominada CHI (Controller Host Interface) que por sua vez está dividida em dois blocos lógicos de acesso: um para configuração e controle de operação do protocolo outro para armazenamento temporário de mensagens. Para transmissão não existem filas, ou seja, uma nova mensagem substitui a mais antiga ao passo que na recepção, filas são opcionais, podendo ser habilitadas de acordo com a necessidade da aplicação. No bloco de configuração e controle, estão disponíveis para leitura e escrita os

parâmetros internos de operação da plataforma. Para cada canal, o host deve informar o tamanho do ciclo de cluster, número de slots em cada segmento, entre outros. Logo, não existe o conceito de MEDL uma vez que toda a configuração referente a atividade de rede provém do host. O projeto do código da aplicação e a configuração da plataforma não são independentes.

4.2.1 Modelo de Comunicação

O controle de acesso no FlexRay é construído sobre uma base de tempo global sincronizada e a comunicação é organizada em ciclos básicos de acesso ao meio físico de mesma duração que se repetem ao longo do tempo.

Cada ciclo básico possui uma janela de tempo dedicada a transmissão síncrona, chamada de **segmento estático**, e outra janela para transmissão assíncrona, chamada de **segmento dinâmico**, conforme ilustra a Figura 4.7.

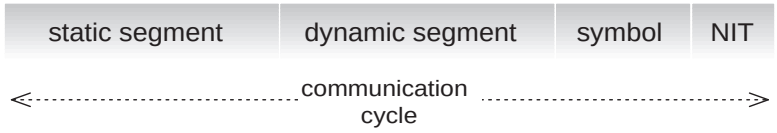

Figura 4.7: Particionamento do Ciclo de Comunicação

Durante o segmento estático (Figura 4.8) os quadros são transmitidos dentro de slots de tempo dedicados com **duração fixa** utilizando multiplexagem no tempo (TDMA), enquanto que no segmento dinâmico o tempo de acesso é adaptativo, i.e. varia conforme a demanda.

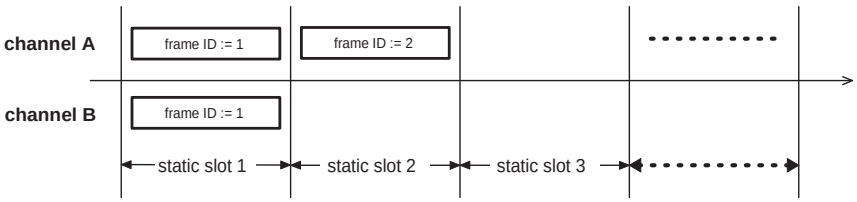

Figura 4.8: Segmento estático.

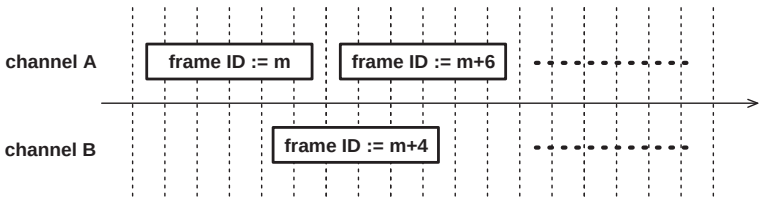

Figura 4.9: Segmento dinâmico.

O controle de acesso no segmento dinâmico (Figura 4.9) utiliza um mecanismo do tipo FTDMA (Flexible Time Division Multiple Access) no qual o tempo de comunicação é particionado em slots de pequeno tamanho denominados **minislots** de tamanho fixo. O número de minislots por ciclo básico varia durante a execução do protocolo em função da demanda por comunicação no segmento dinâmico.

Caso a aplicação não exija suporte para comunicação assíncrona, a quantidade de minislots pode ser zero, suprimindo assim todo o segmento dinâmico e tornando o sistema essencialmente time-triggered.

O controle de acesso FTDMA ocorre da seguinte forma: Para cada um dos minislots existe um identificador único, i.e. um número inteiro que identifica o minislot de maneira exclusiva. Quando a base de tempo acusa o início de um novo minislot **m**, o controlador transmite um quadro se e somente se as 3 condições seguintes forem satisfeitas:

1. Existe uma mensagem assíncrona a ser transmitida.
2. O identificador atribuído ao nodo coincide com o valor atual do contador de minislots gerenciado pelo controlador de rede.
3. O tempo de transmissão do quadro é menor do que o tempo que resta para o término do segmento dinâmico atual.

Para gerenciar as transmissões cada nodo possui um contador de slot/minislot dentro do ciclo (um para cada canal no caso de utilização de uma estrutura replicada), que é iniciado com o valor **m**=1 no início do ciclo de comunicação e incrementado sempre no final de cada slot/minislot de tempo.

Até o final do segmento estático os contadores dos canais (A e/ou B) estão sincronizados. A partir do início do segmento dinâmico as contagens se tornam independentes uma vez que minislots podem tem diferentes tamanhos.

A relação entre identificador e nodo é de N:1, ou seja, um ou mais identificadores podem ser atribuídos a um mesmo nodo porém, o mesmo identificador nunca pode ser atribuído a dois ou mais nodo distintos para garantir que colisões não ocorram dentro dos segmentos estático e dinâmico.

No momento em que o início de uma transmissão é percebido pelos demais nodos da rede, os contadores de minislots de seus controladores ficam paralisados até que o quadro seja transmitido por completo. Em seguida, os contadores incrementam e um novo minislot (**m+x**) é iniciado.

4.2.2 Criação da Base de Tempo

A base de tempo de um sistema FlexRay é criada através de um mecanismo distribuído tolerante à falhas que tem por objetivo trazer os relógios de todos os nodos envolvidos suficiente próximos para que então o algoritmo de sincronização descrito no item 4.2.3 possa funcionar. Durante esta fase inicial o controle de acesso é obviamente diferenciado uma vez que colisões podem ocorrer.

O procedimento é dividido em duas fases: a primeira chamada de **wakeup** e a segunda **startup**. Na primeira fase, o sistema é ativado pela transmissão de uma seqüencia específica de bits denominada de **padrão de wakeup** (WP). A iniciativa da transmissão do WP vem do host e pode surgir de qualquer nodo. Ao receberem este padrão os demais nodos transicionam simultaneamente para a segunda fase quando é dado início ao procedimento de criação da base de tempo global propriamente dito.

Devido ao fato de não existir um entendimento global do tempo na primeira fase, dois ou mais nodos podem iniciar a transmissão de um padrão ao mesmo tempo resultando em colisão. A especificação do protocolo aponta que, em caso de colisão, o meio físico deve garantir que o sinal resultante ainda possa ser interpretado corretamente pelos nós receptores. Assume-se também que os dispositivos que implementa a camada física (transdutores de sinal) sejam capazes de identificar quando a colisão acontece.

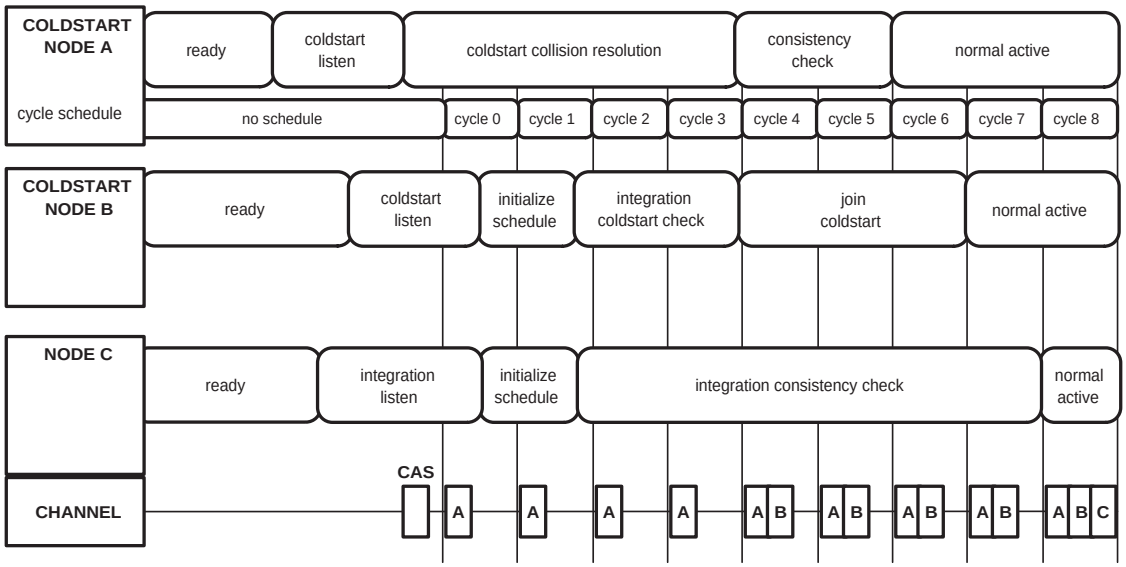

Figura 4.10: Procedimento de entrada a frio no FlexRay.

Na segunda fase, somente alguns nodos (1 ou mais) são autorizados a tomar a iniciativa de criar uma nova base de tempo: procedimento também chamado de entrada a frio (coldstart) tal como na TTA.

O procedimento de entrada a frio mostrado na Figura 4.10 se resume nas seguintes etapas:

1. Um nodo que inicia uma entrada a frio primeiro observa o meio físico para verificar se já não existe comunicação em curso. Caso o meio esteja em silêncio, i.e. sem atividade, o nodo transmite um quadro de controle denominado de CAS (Collision Avoidance Symbol). A transmissão do CAS também está sujeita a colisões logo, as seguintes situações são possíveis:

   (a) Colisão entre CASs: Quando dois ou mais nodos iniciam o procedimento de entrada a frio ao mesmo tempo. Neste caso os nodos que colidiram voltam à fase de observação do meio físico (coldstart listen).

   (b) Nenhuma colisão: O meio físico está em silêncio – o nodo que transmitiu o CAS com sucesso é identificado como **líder**.

2. Nos quatro primeiros ciclos após a transmissão bem sucedida do CAS somente o líder transmite quadros de sincronização com informação de tempo explícita. Os primeiros nodos que se integram ao ritmo de transmissão do líder são aqueles que também têm autorização para iniciar uma entrada a frio, em seguida todos os outros.

3. A partir do quarto ciclo os primeiros nodos que se integram também começam a transmitir quadros de sincronização após o correto recebimento de 4 quadros do líder – condição necessária para trazer as bases de tempo para dentro dos limite de precisão. As informações de tempo explícitas e implícitas dos 2 primeiros quadros são utilizadas para efetuar o primeiro ajuste nos relógios locais. Os outros 2 quadros fornecem informação de entrada para o cálculo do primeiro fator de correção. Caso

nenhum erro ou omissão de quadros por parte do líder ocorra, os primeiros nodos iniciam a transmissão de seus quadros de sincronização.

4. Durante a entrada a frio o nodo líder aguarda pela transmissão de pelo menos 2 quadros para verificar a consistência da base de tempo, i.e. as leituras extraídas da transmissão destes quadros devem ser válidas. Se nenhuma anomalia é detectada então o líder entra em regime normal de operação e inicia a transmitir quadros com mensagens provenientes da camada da aplicação.

5. Os demais nodos permanecem em silêncio até que pelo menos dois quadros de sincronização sejam transmitidos em dois ciclos consecutivos. A partir das leituras de tempo realizadas, seus relógios adotam a base de tempo do líder e entram em regime normal de operação.

4.2.3 Sincronização de Relógios

O algoritmo de sincronização de relógios tem a função de corrigir diferenças nas bases de tempo locais que surgem no decorrer da operação do sistema e que são resultantes de flutuações de temperatura, tensão da fonte de alimentação e tolerância dos dispositivos osciladores (FLEXRAY, 2004). A base de tempo global – estabelecida quando todos os relógios estão dentro dos limites de precisão – é gerenciada de forma hierárquica de acordo com o esquema mostrado na Figura 4.11.

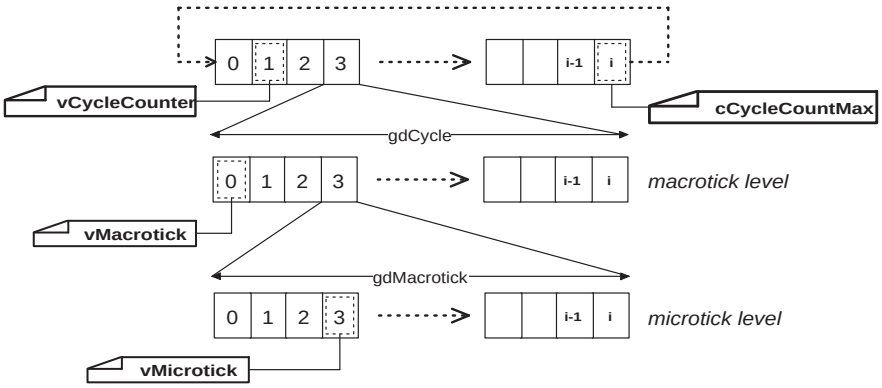

Figura 4.11: Hierarquia da base de tempo.

O nível de microtick é a fonte de tempo primária extraída diretamente do dispositivo oscilador da plataforma. Logo acima está definido o nível de macrotick – granularidade mínima de tempo em um sistema distribuído FlexRay. Todas as ações relevantes são agendadas e/ou computadas em unidades inteiras de macroticks. No nível mais alto da hierarquia estão os ciclos de comunicação do sistema, numerados de 0→cCycleCountMax. O tamanho do ciclo de comunicação (gdCycle) é uma constante para todo o cluster e deve ser configurado com um número inteiro (fixo) de macroticks.

O protocolo utiliza um mecanismo combinado de correção de freqüência (amortização) e de fase (método discreto). O processo de correção de freqüência é executado durante todo o tempo de operação normal do sistema enquanto que a correção de fase é feita a cada dois ciclos, durante a fase de NIT (Network Idle Time) conforme mostra a Figura 4.12.

As informações de entrada para a função de convergência são obtidas através de leituras indiretas de relógios, todas efetuadas durante o segmento estático. Para que seja

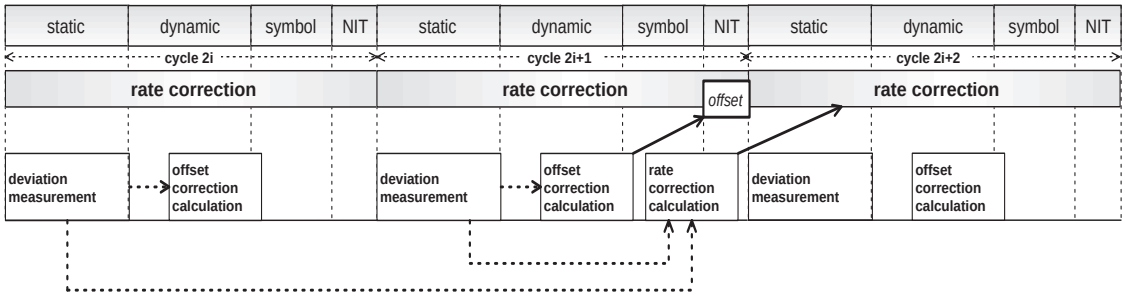

Figura 4.12: Etapas do Processo de Sincronização

possível extrair os desvios relativos, o procedimento de obtenção das leituras faz uso de pontos de referência de tempo escolhidas em tempo de projeto. Dentro do segmento estático, o início de uma transmissão i coincide sempre com um AT_i (Action Time de i) logo, o valor de AT_i serve para representar o instante esperado do início da transmissão i no lado dos receptores.

O instante de transmissão efetivo é obtido a partir do esquema mostrado na Figura 4.13 que mostra a seqüencia padrão de início de quadro.

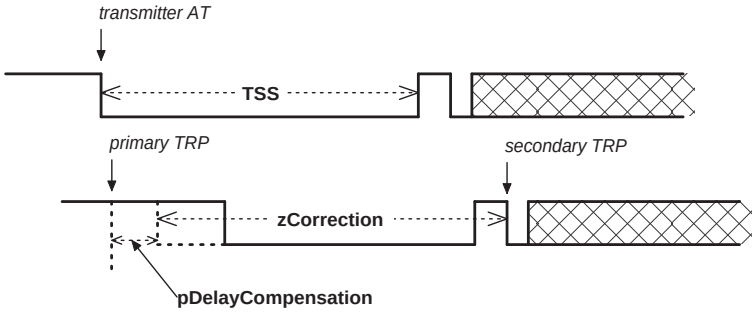

Figura 4.13: Sequência Binária de Início de Quadro

O intervalo indicado por TSS (Transmission Start Sequence) representa uma proteção contra truncagem visto que alguns bits do início do quadro podem ser sucumbidos pelo acoplador ótico, quando a topologia é do tipo estrela. Após a TSS o início de quadro possui um evento bem definido (transição $1 \rightarrow 0$) quando o receptor deve amostrar a sua base de tempo local e armazenar este valor em uma variável chamada de **referência secundária** (*secondary TRP*).

O AT do transmissor (*transmitter AT*) é então inferido a partir da TRP secundária e dos parâmetros **zCorrection** e **pDelayCompensation**, que compensam respectivamente o tempo gasto para transmissão da TSS e o atraso de propagação do sinal no meio. Subtraindo os valores de **zCorrection** e **pDelayCompensation** da TRP secundária chega-se ao valor da **referência primária** (*primary TRP*), que representa uma estimativa da base de tempo do nodo transmissor. A partir de cada referência primária é avaliado então o desvio deste valor em relação a AT, i.e. o instante esperado.

Os valores dos desvios são representados em unidades de microtick. Na medida em que vão sendo obtidos durante os segmento estático eles vão sendo armazenados em uma estrutura de dados interna do controlador chamada de **DevTable**. A cada início de ciclo uma parte desta tabela é apagada. No inicio de um ciclo ímpar, todos os valores provenientes do último ciclo ímpar são apagados e, no início de um ciclo par, são apagados os

valores obtidos no último ciclo par.

A especificação do protocolo informa que nem todas as transmissões que ocorrem dentro do segmento estático são utilizadas para cálculo de desvios. É necessário que o quadro em questão possua o atributo de quadro SYNC. Ao contrário da TTA onde esta informação é local, no FlexRay está informação vem junto com o quadro.

Após a fase de coleta e armazenamento dos desvios na **DevTable** o protocolo FlexRay executa o algoritmo FTM (WELCH; LYNCH, 1988) para cálculo do fator de correção utilizando os desvios como entrada para a função de convergência. O fator de correção de fase é calculado em todo ciclo de comunicação enquanto que o fator de correção de freqüência é calculado somente nos ciclos ímpares, durante o intervalo de tempo existente entre o término do segmento estático e o início do próximo ciclo conforme indicado na Figura 4.12.

Em função do número de valores de desvio disponíveis, o algoritmo FTM é executado da seguinte forma:

1. Uma constante κ é determinada de acordo com a quantidade de valores de desvios armazenados na **DevTable**. Existem três possíveis valores para κ conforme mostrado na tabela 4.1.

Tabela 4.1: Critério de escolha do parâmetro κ.

number of values	κ
1 - 2	0
3 - 7	1
>7	2

2. Os valores de desvios disponíveis são ordenados e então os κ maiores e também os κ menores são descartados.

3. Dos valores restantes, é escolhido o valor médio como fator de correção.

Para cálculo do fator de correção de fase, os desvios utilizados na execução do FTM devem ser obtidos durante o segmento estático do mesmo ciclo de comunicação. O valor final é um inteiro e indica o número de microticks que o início do próximo ciclo deve ser deslocado. Valores negativos indicam que a NIT deve ser resumido com antecedência enquanto valores positivos indicam que a NIT deve ser prolongado.

Os desvios utilizados no cálculo do fator de correção de freqüência são diferenças de tempo entre desvios extraídos de dois ciclos de comunicação consecutivos. Neste caso o valor de correção é um inteiro que indica o número de microticks que deve ser inserido ou retirado de um ciclo de comunicação.

O interessante é que em ambos os casos, o fator de correção é um inteiro que expressa divergências no nível mais alto da hierarquia de tempo, em termos de duração e deslocamento de fase dos ciclos de comunicação entre as bases de tempo distribuídas. Contudo, a única variável afetada pelo processo de sincronização é o contador de macroticks (**vMacrotick**). A especificação do protocolo define um processo de controle denominado de MGP (Macrotick Generation Process) que tem como objetivo distribuir os fatores de correção de maneira uniforme dentro dos ciclos. As correções necessárias são aplicadas

de maneira contínua e gradual. O fator de correção de freqüência é aplicado ao longo de dois ciclos de comunicação enquanto que o fator de fase é aplicado durante o intervalo NIT, a cada dois ciclos.

De acordo com a documentação, uma estimativa da precisão máxima de um sistema FlexRay pode ser obtida a partir da seguinte fórmula:

$$vWorstCasePrecision = \beta \cdot gdMaxMicrotick + 2 \cdot \Delta_{pdmax}$$

onde *gdMaxMicrotick* provém do nó mais lento conforme já mencionado, Δ_{pdmax} é o atráso máximo de propagação no meio físico que por sua vez é dado por:

$$\Delta_{pdmax} = 2 \cdot \Delta_{tx} + l \cdot \Delta_{line} + N_{hubs} \cdot \Delta_{hub}$$

e β é um fator multiplicativo igual a 134 que representa a influência do algoritmo de sincronização utilizado.

Caso a topologia em questão seja do tipo estrela, N_{hubs} e Δ_{hub} representam o número e o atraso em cada acoplador ótico. Por fim, Δ_{tx} é o atraso dos transdutores, l é a distância máxima entre dois nós e Δ_{line} o atraso de propagação por unidade de distância (metros).

Apesar de não fornecer nenhum desenvolvimento matemático, a especificação do FlexRay apresenta valores ditos como típicos para cada uma destas variáveis e chega a uma precisão de 11.7us. Contudo, este valor é calculado em função do pior caso.

4.3 TTCAN

4.3.1 Protocolo CAN

O CAN, definido pela ISO-11898, é um protocolo para comunicação assíncrona de dados através de um barramento serial. Originalmente o CAN foi criado pela Bosch GmbH no intuito de solucionar o problema da crescente demanda por comunicação entre módulos eletrônicos nos automóveis que até então eram conectados por meio de cabeamento dedicado. Com o tempo o protocolo ganhou alta popularidade também em aplicações industriais, equipamento médico e hospitalar, automação residencial, entre outros.

No CAN somente as camadas 1 e 2 do modelo OSI são definidas. Na camada 2, o controle de acesso é do tipo CSMA/CD–AMP, onde o termo AMP (Arbitration on Message Priority) indica que colisões são resolvidas por um mecanismo de arbitração bit a bit de natureza não-destrutiva baseado em prioridades, conforme mostra a Figura 4.14. Os primeiros 11 bits de cada quadro (formato standard) formam um identificador exclusivo que determina sua prioridade. Quanto menor o valor do identificador, mais alta é a prioridade de transmissão deste quadro.

Para o funcionamento do mecanismo de arbitração não-destrutivo são definidos na camada 1 os níveis recessivo – valor binário ’1’, e dominante – valor binário ’0’. Um nível dominante prevalece sobre um recessivo, o que significa dizer que se ao menos um controlador enviar um bit dominante então o nível de todo barramento será dominante. No momento da transmissão de uma mensagem os controladores monitoram o nível do barramento e o controle de acesso se dá da seguinte forma:

1. Quando o controlador transmite um bit recessivo, o nível lógico do barramento é lido e comparado no mesmo tempo de bit.

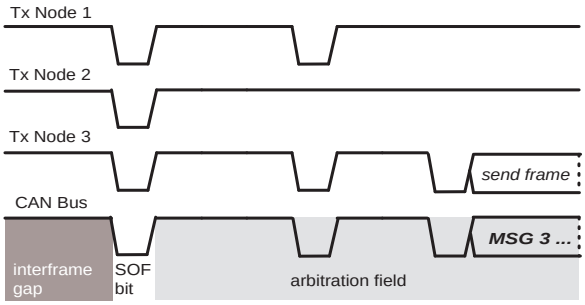

Figura 4.14: Controle de acesso não-destrutivo.

2. Caso o nível lido seja dominante então o controlador suspende a transmissão e se torna receptor da mensagem.

Não existe conceito de endereço físico de módulos, os identificadores estão ligados ao tipo de mensagem e não ao módulo transmissor e/ou receptor. Uma mensagem CAN é transmitida e disponibilizada para todos os módulos da rede ao mesmo tempo, cabendo a cada controlador reconhecer e filtrar mensagens com base nos identificadores.

Uma rede CAN consegue operar em uma velocidade de até 1Mbps, utilizando codificação NRZ, com até 40 módulos conectados a um mesmo barramento. A limitação de velocidade está diretamente ligada ao procedimento de arbitração o qual exige que o tempo de bit seja grande o suficiente para que todos os nodos ativos amostrem simultaneamente o mesmo bit.

Apesar da velocidade de comunicação estar limitada a 1Mbps, a taxa efetiva de transmissão de pequenos dados (variáveis de processo, comandos de disparo, etc) é alta pelo fato de um quadro possuir aproximadamente apenas 34 bits de controle, incluindo CRC de 15 bits. Contudo, o protocolo CAN apresenta deficiências na medida que erros ocorrem com maior freqüência. O determinismo do sistema é afetado pelos mecanismos de sinalização global de erros isolados e retransmissão automática de quadros previstos na norma ISO-11898.

Outro problema do CAN é o alto jitter associado a transmissão periódica de mensagens, principalmente quando a prioridade do identificador é baixa. Sendo de característica assíncrona, uma tentativa de acesso ao barramento acontece de forma imprevisível. Por esta razão o CAN é classificado como um protocolo event-triggered (ALBERT, 2004). Caso dois ou mais módulos tentem transmitir uma mensagem ao mesmo tempo, a mensagem de maior prioridade será enviada primeiro e as demais aguardam próxima tentativa de transmissão. Com o aumento da carga da rede mensagens de menor prioridade sofrem atrasos cada vez maiores.

Apesar destes fatores limitantes em aplicações críticas, o protocolo CAN original representa uma solução extremamente eficiente até hoje e largamente utilizada na indústria.

4.3.2 Extensão Time-Triggered

O TTCAN (Time-Triggered Communication on CAN) foi desenvolvido também pela Bosch, focando em aplicações de controle em sistemas automotivos embarcados, mais especificamente sistemas por-fio (FUHRER et al., 2000), as quais requerem uma plataforma de comunicação TT para melhores resultados em termos de precisão e robustez. A norma internacional ISO11898–4 especifica o TTCAN como uma extensão do protocolo CAN original, que pode ser implementado utilizando controladores de rede já existentes

comercialmente.

Como o nome já indica, um sistema TTCAN segue o modelo time-triggered de comunicação. Durante a fase de projeto define-se uma tabela de escalonamento denominada **matriz do sistema**, que representa o comportamento global da comunicação indicando todos os instantes de início e fim da transmissão de mensagens periódicas, expressos em unidades de tempo de relógios locais que se mantém sincronizados. A matriz do sistema é formada por uma seqüencia de ciclos básicos de comunicação dentro dos quais as mensagens são transmitidas em janelas de tempo de **tamanho variável** segundo um esquema TDMA, conforme mostra a Figura 4.15. Retransmissões automáticas definidas na norma do procotolo CAN original devem ser desativadas, caso contrário uma falha transiente na rede poderia causar uma retransmissão que ultrapassaria os limites de tempo de uma determinada janela.

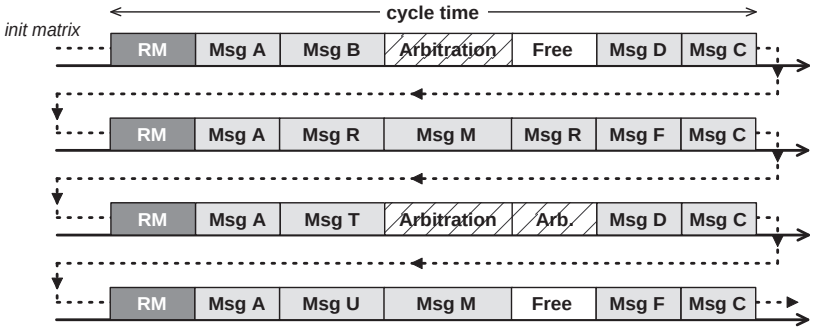

Figura 4.15: Exemplo de uma matrix de comunicação.

O início de cada novo ciclo básico é marcado pela transmissão da RM (Reference Message) originada em um nodo previamente configurado como mestre. A execução de todos os ciclos que formam a matriz do sistema tem uma característica repetitiva, ou seja, ao final da transmissão da última mensagem do último ciclo básico, uma mensagem de referência marca novamente o início da matriz. Dentro de cada janela, um nodo pode transmitir uma mensagem periódica agendada para aquele instante ou uma mensagem aperiódica, em resposta a um evento assíncrono ocorrido internamente ou externamente ao sistema. Janelas alocadas para transmissão de mensagens periódicas são chamadas de janelas exclusivas. Nestas, o instante de transmissão da mensagem deve coincidir com o início da janela e o acesso ocorre, como o nome já diz, de forma exclusiva.

Janelas para transmissão de mensagens aperiódicas são chamadas de janelas de arbitração, cuja principal diferença é o fato de que mais de um nodo pode ter a permissão de transmitir em qualquer instante dentro de seus limites de tempo. A única restrição é que uma transmissão só pode ser iniciada se o tempo de transmissão da respectiva mensagem for menor do que o tempo que ainda resta até o início da próxima janela. Em caso de múltiplas tentativas de transmissão na mesma janela de arbitração, o mecanismo original do protocolo entra em ação fazendo com que mensagens de maior prioridade sejam transmitidas primeiro.

Uma característica particular do TTCAN é que o conhecimento da matrix de sistema em cada nodo é restrita, isto é, um determinado nodo é configurado somente com informações da matrix que se referem a transmissão das mensagens de seu interesse. Neste caso, uma possível vantagem é que, para pequenas mudanças na matrix do sistema, somente alguns nodos precisam ser reconfigurados.

Apesar de ter como principal objetivo o desenvolvimento de aplicações críticas, a es-

pecificação do TTCAN não cobre todos os requisitos de segurança relacionados com sistemas distribuídos embarcados, como por exemplo a redundância de componentes visto. Os nodos do sistema se comunicam através de um único barramento serial. Não existe suporte para a replicação do barramento. Caso a replicação seja indispensável, fica a cargo do projetista a implementação de mecanismos de coordenação de execução e votação de valores.

O protocolo prevê a replicação do nodo mestre. É possível configurar um ou mais nodos como mestres de sombra. Em caso de falha no nodo mestre corrente, o mestre sombra detecta a ausência da RM e pode então tomar a iniciativa de se tornar o novo mestre corrente, assumindo a função de transmitir a RM em seu lugar. Se existem dois ou mais mestres de sombra, deve-se garantir que cada um deles possua seu próprio identificador. Quando todos detectarem a ausência da RM ao mesmo tempo, o mecanismo de arbitração não-destrutivo do CAN original entra em ação garantindo que a RM do mestre de sombra que possui o identificador menor seja transmitida e recebida corretamente por todos os nodos, inclusive os mestres de sombra que não conseguiram transmitir suas RMs.

4.3.3 Sincronização de Relógios

Em um sistema TT-CAN, a base de tempo é **centralizada** uma vez que todos os componentes mantém seus relógios sincronizados a base de tempo do nodo mestre através de mecanismos de correção de fase (correção discreta) e de freqüência (amortização). Os fatores de correção são calculados essencialmente a partir de leituras indiretas e diretas do relógio do mestre. A criação inicial da base de tempo é função única e exclusiva do nodo mestre.

Sempre ao final de cada ciclo básico, os nodos entram em estado de espera pela mensagem de referência. Ao detectar o início de uma transmissão, ou seja, o primeiro bit de uma mensagem CAN, o receptor captura o valor que seu relógio local acusam neste instante. No lado do mestre, o valor do relógio no momento do início da transmissão é incluído no corpo mensagem caracterizando um mecanismo de leitura **explícita**. O fator de correção de fase é então calculado da seguinte forma:

$$\Lambda = C_j - C_i$$

onde C_j é o valor do relógio do receptor e C_i o relógio do mestre que foi transmitido explicitamente no corpo da mensagem. A partir de então Λ unidades de tempo são subtraídas ou adicionadas das janelas de maneira não-uniforme. Quanto mais próximas do fim do ciclo, as janelas sofrem maiores alterações.

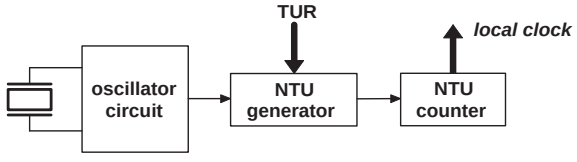

Figura 4.16: Base de tempo local de um nodo TTCAN.

O mecanismo de correção de freqüência atua na geração da unidade de tempo básica do sistema, a NTU (Network Time Unit), cuja granularidade mínima é função direta da precisão da base de tempo global. O tamanho de uma NTU (em ciclos de máquina) depende do valor da TUR (Time Unit Ratio) que é determinado da seguinte forma:

$$\text{TUR} = \frac{C_i^k - C_i^{k-1}}{C_j^k - C_j^{k-1}}$$

onde C_i^k e C_i^{k-1} são instantes de transmissão da RM associados a dois ciclos consecutivos do ponto de vista do receptor enquanto que C_j^k e C_j^{k-1} são os instantes de transmissão da RM do ponto de vista do nodo mestre.

4.4 SAFEbus

O SAFEbus, definido na norma ARINC659, é uma plataforma de comunicação utilizada em aeronaves. Originalmente, o SAFEbus foi proposto pela Honeywell com o objetivo de proporcionar uma solução altamente robusta, que oferecesse todo o suporte necessário ao projeto integrado de funções para o sistema de controle de informações da aeronave Boeing 777 (HOYME; DRISCOLL, 1993).

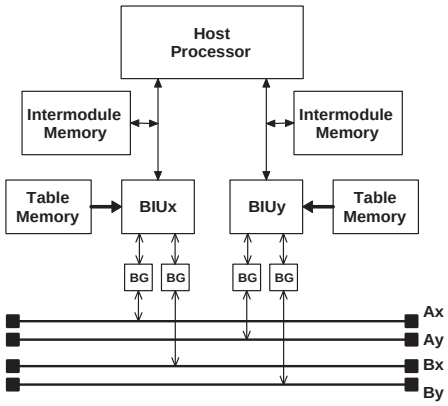

Figura 4.17: Organização da arquitetura SAFEbus.

A organização da plataforma SAFEbus é mostrada na Figura 4.17. O meio físico é redundante, formado por quatro barramentos protegidos por dispositivos de guarda (*bus guardians*). Cada barramento possui linhas dedicadas para transmissão do clock e dois bits de dados a uma taxa de 30Mbps.

O controle acesso aos barramentos é implementado dentro dos BIUs (*bus interface units*). O BIUy é conectado aos barramentos Ay e By, enquanto que o BIUx se conecta a Ax e Bx, fazendo com que a infraestrutura de comunicação seja capaz de mascarar falhas que porventura ocorram em Ax/Ay ou Bx/By. Mensagens recebidas pelo par de barramentos Ax/Ay ou Bx/By são comparadas bit a bit e são aceitas somente se forem iguais.

O protocolo de comunicação opera sobre uma base de tempo global de forma que todos os nós tenham um entendimento comum dos instantes de início de cada janela de acesso. A criação da base de tempo, ou seja, a sincronização inicial dos relógios é obtida pelo envio de pulsos na linha de clock. Eventuais conflitos entre pulsos transmitidos por diferentes nodos são resolvidos no nível elétrico pelos transdutores que fazem do meio físico um OU lógico. Em regime normal de operação, todos os nodos transmitem pulsos de sincronização em instantes definidos na Table Memory (TM). O sinal resultante destes pulsos, percebido da mesma forma por todos os nodos, marca uma nova referência de tempo global, sobre a qual todos os BIUs do sistema se alinham fazendo uma dilatação ou contração do espaço entre mensagens. Observa-se que a base de tempo é descentralizada uma vez que o mecanismo de sincronização não depende de nenhum nodo em específico.

O funcionamento do protocolo em tempo de execução é totalmente estático o que garante o particionamento de funções no tempo e no espaço. Todo o comportamento da comunicação é definido na forma de comandos de transmissão e recepção que são armazenados na TM durante a fase de projeto. Além de comandos, a TM também contém ponteiros para busca e armazenamento de mensagens na memória de interface com o host que recebe o nome de Intermodule Memory (IM). O host é totalmente desacoplado da TM garantindo que falhas no host não se propaguem pelos barramentos na forma de mensagens espúrias e afetem outros nodos.

Não existe arbitração de mensagens nos barramentos visto que cada comando de transmissão é executado em uma janela de tempo dedicada. Os comandos são agrupados em diferentes "planos" formando uma seqüencia repetitiva de ações de transmissão, recepção ou de espera. Mensagens carregam somente dados. Toda informação de controle necessária para o processamento de cada mensagem é lida diretamente da TM.

Existem dois tipos de transmissão: básica e mestre-sombra. O tipo de transmissão que ocorre em cada janela é especificado na TM. Na transmissão básica, um nó tenta transmitir a mesma mensagem nos 4 barramentos ao mesmo tempo. No caso da transmissão tipo mestre-sombra, um nodo previamente definido como mestre é quem deve transmitir a mensagem. Se, após Λ unidades de tempo nada for transmitido na respectiva janela, então um outro nodo previamente definido como sombra inicia a transmissão. No host, o software da aplicação acompanha a comunicação com o uso freqüente de interrupções disparadas pelos BIUs caracterizando um comportamento essencialmente time-triggered, onde as ações são disparadas na medida que uma base de tempo global evolui.

4.5 DACAPO

A arquitetura DACAPO (ROSTAMZADEH et al., 1995) foi desenvolvida na Chalmers University of Technology em colaboração com a indústria automotiva européia, no intuito de oferecer uma solução flexível, econômica e de fácil instalação, com a possibilidade de integrar funções de controle distribuídas sobre uma mesma plataforma.

A organização interna de um nodo na arquitetura DACAPO é mostrada na Figura 4.18. Observa-se a utilização de blocos replicados (indicados pelas diferentes tonalidades de cinza) dispostos em uma estrutura simétrica dual, formada por dois grupos de blocos funcionais idênticos. A comunicação interna entre os blocos de um mesmo grupo ocorre pela transferência direta de dados através de barramentos paralelos enquanto que a comunicação entre os grupos replicados é do tipo memória compartilhada. A infraestrutura física para comunicação serial entre nodos também é replicada, formada por dois barramentos seriais redundantes.

Assume-se que todos os componentes possuem a característica *fail–silent*, isto é, em caso de falha não fornecem nenhum resultado nem interferem no funcionamento de outros. A CMU (*Communication Unit*) possui duas GCIs (*Global Communication Interfaces*) que são responsáveis pelo controle de acesso aos barramentos, serialização e paralelização de dados e sincronização de relógios, baseada na correção da base de tempo pelo método Daisy Chain (LONN, 1999b), onde todos os nodos reajustam suas bases de tempo a cada mensagem transmitida. O componente EC (*Embedded Controller*) possui duas interfaces de entrada e saída que se comunicam diretamente com sensores e atuadores, dois PEs (*Processing Elements*) que executam as tarefas de controle, duas SMs (*Stable Memories*) utilizadas na comunicação entre os PEs, e uma unidade central de controle chamada CCU (*Cross Coupling Unit*). São previstas duas fontes de alimentação por nodo, uma

fornecendo energia aos componentes representados pela tonalidade cinza mais escura na Figura 4.18 e a outra aos componentes indicados pela tonalidade de cinza claro.

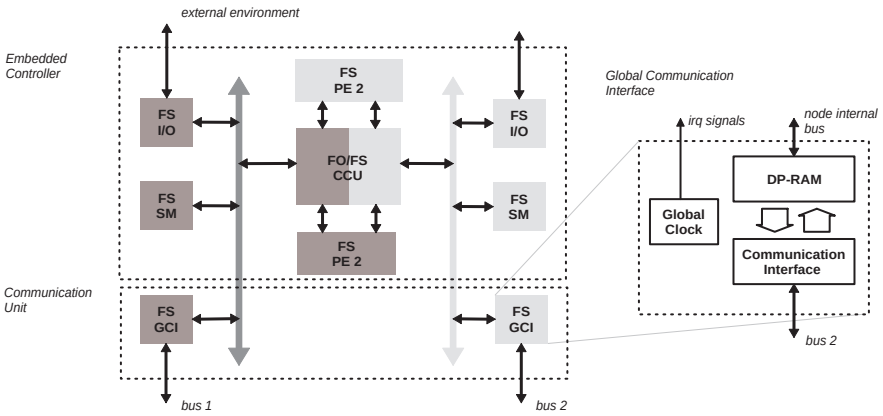

Figura 4.18: Organização Interna de um nodo na DACAPO.

Na DACAPO, o escalonamento das tarefas e do tráfego de mensagens relacionadas a tarefas de controle é definido a priori, caracterizando um paradigma de processamento e comunicação estritamente time-triggered. As mensagens são transmitidas fazendo uma multiplexagem do tempo de acesso conforme uma política TDMA. Cada uma carrega até 8 bytes de dados em quadros de aproximadamente 100 bits. O valor corrente do contador de ciclo e o identificador do nodo transmissor são inseridos no corpo de todas as mensagens para permitir rápida integração de novos nodos ou de nodos que se recuperam de falhas transitórias.

Durante a fase de projeto é definido o chamado **bloco de execução** onde são especificados todos os instantes de início, término ou preempção das tarefas de controle. Em regime normal de operação, cada execução do bloco é chamada de ciclo de sistema (SC). O SC é subdividido por sua vez em ciclos de comunicação (CCs) dentro dos quais as tarefas são escalonadas. Ao final do último CC o sistema volta ao início do SC executando o primeiro CC novamente. O início de cada CC é informado aos PEs por sinais de interrupção que provém de uma base de tempo global disponível nas GCIs. Dentro dos limites de tempo de um CC os nodos transmitem suas mensagens em janelas de tempo dedicadas. Janelas vazias resultantes do escalonamento de mensagens podem ser utilizadas para transmissão de mensagens aperiódicas.

De acordo com os autores, as principais vantagens relacionadas à definição estática do comportamento do sistema são as seguintes:

1. Projeto mais simples, em razão da maior facilidade de verificar o cumprimento de requisitos de tempo antes da instalação do sistema.

2. O efeito de uma falha transitória é limitado no tempo. Cada variável de controle que é transmitida pela rede é associada a uma flag de controle do tipo *stale*/*updated*. Lembrando que em sistemas time-triggered valores novos sobrescrevem valores antigos, deve-se prover mecanismos para informar a aplicação quando uma determinada variável está desatualizada. No lado do receptor, se uma mensagem não é recebida no instante previsto então a flag de controle é automaticamente setada para *stale*. No lado do transmissor, se o PE falha ao fornecer um valor antes do instante de transmissão da respectiva mensagem, então o ultimo resultado válido é

enviado junto com a flag setada para *stale*. A aplicação pode então decidir disparar uma rotina de emergência que utiliza, por exemplo, uma outra variável ou o último resultado calculado.

3. O conhecimento prévio dos instantes de transmissão das mensagens permite uma detecção mais rápida de erros.

4. Latências são reduzidas e fixas. Para minizar os atrasos no sistema, toda tarefa deve iniciar logo após seus valores de entrada terem sido recebidos. Da mesma forma, seus resultados devem ser escalonados para transmissão imediatamente após terem sido calculados.

Na arquitetura DACAPO, os nodos têm um comportamento do tipo FO/FS (*fail–operational* → *fail–silent*), ou seja, um nodo continua operando corretamente mesmo com uma falha permanente e, caso uma segunda falha permanente ocorra, o nodo assume um comportamento silencioso. Mecanismos de detecção de erros são implementados em software e em hardware para garantir um rápido isolamento de um componente defeituoso. Cada PE é composto por dois processadores totalmente sincronizados, isto é, possuem um relógio comum. Resultados calculados por estes processadores são sempre comparados no objetivo de detectar inconsistências. Quando alguma comparação acusa uma falha, o PE imediatamente cessa de se comunicar com os demais componentes.

Em regime normal de operação, os dois PEs ficam se alternando na execução das tarefas e cálculo de resultados. O instante de chaveamento é o início de um CC. Enquanto um dos PEs executa um CC, o estado das tarefas vai sendo armazenado em uma das SMs. No momento do chaveamento, o outro PE assume o controle do nodo e continua a execução do SC com base nas informações disponibilizadas na SM pelo outro PE. Quando um PE falha, o outro imediatamente assume controle total da execução das tarefas.

4.6 Análise Comparativa

A tabela 4.2 mostra uma análise comparativa das plataformas apresentadas neste capítulo, tomando como base de comparação os aspectos considerados mais relevantes no projeto de sistemas críticos. Todos operam em TDMA e garantem particionamento temporal, contudo observa-se diferenças significativas no que se refere a características arquiteturais.

Tabela 4.2: Avaliação comparativa de plataformas.

	FlexRay	**TTCAN**	**TTP/C**	**DACAPO**	**SAFEbus**
BASE DE TEMPO	DISTRIBUÍDA	CENTRALIZADA	DISTRIBUÍDA	DISTRIBUÍDA	DISTRIBUÍDA
PRECISÃO	1 μs	1 μs	1 μs	–	33 ns
TAMANHO DE SLOT	FIXO	VARIÁVEL	FIXO	FIXO	VARIÁVEL
TRÁFEGO ASSÍNCRONO	INTRÍNSECO	INTRÍNSECO	LIMITADO	INEXISTENTE	INEXISTENTE
TAXA DE TRANSMISSÃO	10 Mbps	1 Mbps	20 Mbps	–	15 Mbps
VAZÃO EFETIVA	0.20	0.28	0.17	0.24	1
TOLERÂNCIA À FALHAS	INTRÍNSECO	LIMITADO	INTRÍNSECO	INTRÍNSECO	INTRÍNSECO
CUSTO	MÉDIO	BAIXO	MÉDIO	ELEVADO	ELEVADO

A base de tempo global pode ser distribuída, quando o algoritmo de sincronização não depende de um nodo específico, caso contrário é dita centralizada. O tamanho do slot pode ser fixo ou variável. Esta característica é importante conhecer quando a carga de comunicação da aplicação é elevada. Em protocolos TDMA com slots de tamanho fixo geralmente existe um desperdício do tempo de transmissão em vista do fato deste parâmetro ser determinado pelo pior caso, isto é, maior quadro a ser transmitido.

No que se refere ao suporte para transmissão de mensagens assíncronas a plataforma pode ou não oferecer este serviço. Plataformas que não suportam tráfego assíncrono sofrem de uma falta de capacidade de configuração e diagnóstico, além de não serem capazes de tratar eventos externos tais como disparos de alarme.

A vazão efetiva é definida como sendo a relação entre o tamanho de uma mensagem de 16 bits e o tamanho de um quadro necessário para transmití-la, considerando sempre o pior caso, ou seja, o quadro de maior tamanho (mais completo) do protocolo.

Tolerância à falhas se refere ao suporte que a plataforma oferece para o uso de técnicas tais como redundância ativa, unidades de reserva, entre outros. Na maioria dos casos existe suporte intrínseco, ou seja, controladores de rede são construídos utilizando redundância de elementos funcionais, replicação de barramentos, dispositivos de guarda, etc. A única exceção é o TTCAN, que oferece suporte relativamente limitado, principalmente em vista do fato da sua base de tempo ser do tipo centralizada.

Por último considera-se o custo de implementação de um sistema distribuído utilizando uma solução ou outra. Fatores que influenciam o custo são basicamente o número de canais de comunicação, complexidade do projeto e a utilização de recursos de hardware e software. Considera-se que a DACAPO e a SAFEbus resultam em um projeto mais custoso pela complexidade e alta utilização de recursos de hardware dos controladores de rede, onde praticamente todos os blocos funcionais são replicados.

5 TRABALHO REALIZADO

5.1 Descrição Geral da Plataforma

O trabalho realizado consiste na proposta e implementação de uma plataforma de comunicação para sistemas de controle distribuídos a qual foi dado o nome de CASCA – uma abreviatura para **Communication Architecture for Safety-Critical Applications**. CASCA é resultado do estudo de outras propostas e trabalhos relacionados que foram apresentados nos capítulos 3 e 4. A partir de uma análise dos requisitos de sistemas críticos observou-se que questões relacionadas à performance e consumo energético dos circuitos integrados ficam em segundo plano. Logo, CASCA prioriza determinismo e oferece suporte para o mapeamento de unidades redundantes com um esquema de acesso TDMA construído sobre uma base de tempo global, a qual possibilita o mapeamento e a execução em perfeita sincronia de unidades funcionais redundantes.

CASCA segue um modelo de comunicação time-triggered inspirado na TTA, onde as transmissões têm início em instantes de tempo precisos definidos sobre uma base de tempo global. Funções primitivas de sincronização de relógios e criação da base de tempo foram implementadas sobre o protocolo CAN nativo, na forma de blocos de hardware e software. Sincronização de relógios foi implementada com base na descrição formal do algoritmo Daisy-Chain (LONN, 1999b) ao passo que o método de criação da base de tempo é original. Para isto, foi utilizada a linguagem VHDL na descrição visando a síntese e prototipação imediata em dispositivos FPGA de baixo custo. No total, foram contabilizadas aproximadamente 3200 linhas de código RTL. Algumas funções foram implementadas em software fazendo uso da linguagem C. O código gerado é executado em um processador para uma maior flexibilidade de projeto e para diminuir da utilização de recursos de lógica programável. A opção pelo uso do processador também se deve à possibilidade de poder modificar o comportamento do protocolo após a definição de uma estrutura de hardware final que futuramente seria proposta para síntese em silício.

5.1.1 Ciclo de Vida do Projeto

O projeto da plataforma proposta pode ser divido em 4 principais etapas:

1. Implementação das camadas física e de enlace do protocolo.
2. Implementação de um algoritmo de sincronização de relógios.
3. Implementação de um procedimento de criação da base de tempo global.
4. Verificação funcional e validação em protótipo.

Na primeira etapa, um controlador CAN foi desenvolvido no intuito de oferecer serviços básicos das camadas 1 e 2. Os fatores que influenciaram nesta escolha são apresentados a seguir:

Documentação rica e de fácil acesso:

A especificação do protocolo CAN é aberta, podendo ser utilizada sem custo algum. Além disso, uma vasta documentação técnica é disponibilizada nos sites da IEEE, ACM, BOSCH Semiconductors e CAN in Automation (CiA), na forma de trabalhos científicos, relatórios técnicos, datasheets, entre outros.

Alta popularidade:

O protocolo CAN é hoje de ampla utilização tanto em automação industrial quanto em sistemas embarcados de grande porte. Por isso existe uma alta disponibilidade de dispositivos tais como transdutores e controladores stand-alone, facilmente encontrados no mercado.

Características elétricas favoráveis:

O barramento serial transmite os dados em níveis de tensão diferenciais utilizando codificação do tipo NRZ (Non Return to Zero). Além de dissipar menos potência pelo fato do sinal não transicionar durante todo o tempo de bit, a transmissão diferencial NRZ é menos sensível à interferência eletromagnética e possui baixa emissão de ruído quando comparada a outras técnicas de codificação de sinal, como por exemplo codificação Manchester.

Controle de acesso determinístico:

Conforme já descrito no capítulo 4, o mecanismo de arbitração do protocolo CAN garante que, em caso de colisão, a mensagem com identificador de maior prioridade prevalece sem mesmo perceber que a colisão ocorreu. Descobriu-se posteriormente que esta característica também facilitaria o processo de criação da base de tempo distribuída.

Baixo overhead de controle:

Um quadro no formato padrão contém somente 35 bits de controle [1], proporcionando uma maior eficiência para transmissão de dados com poucos bits, como é o caso de variáveis de processo em sistemas de controle. Este número reduzido de bits de controle implica também numa menor quantidade de recursos de hardware necessários para implementação do controlador de rede.

A Figura 5.1 mostra a arquitetura básica resultante da primeira fase. O bloco ISO11898 formado pelos módulos BSP (Bit Stream Processor) e BTL (Bit Time Logic) implementa as funções básicas de um controlador de rede CAN tais como serialização e paralelização de dados, arbitração, checksum, confirmação em tempo de bit, entre outros. Retransmissões automáticas e geração de quadros de erro previstos no protocolo original foram excluídos para garantir o particionamento temporal. Controladores de rede CAN convencionais funcionam de forma que a cada erro detectado[2] durante o processo de recepção

[1] Desconsiderando stuff bits.

[2] O protocolo CAN possui 5 mecanismos de detecção de erros distintos: *bit error*, *acknowledgment error*, *arbitration error*, *stuff error* e *form error*. Para maiores detalhes consultar a especificação do protocolo (CIA, 1999)

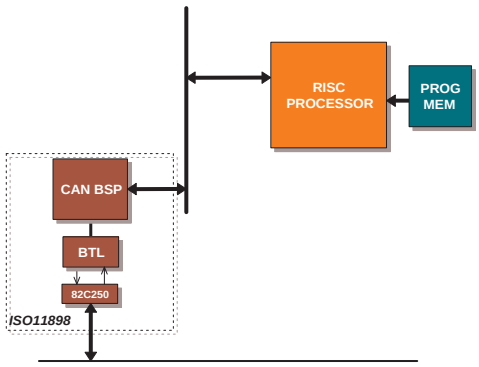

Figura 5.1: Arquitetura básica – protocolo CAN nativo.

ou transmissão uma notificação é gerada em broadcast para todos os outros controladores conectados ao mesmo barramento. Esta notificação recebe o nome de **quadro de erro** (*error frame*) e faz com que todos os receptores abortem e o transmissor inicie automaticamente uma nova transmissão do respectivo quadro. Em um esquema TDMA, o problema das retransmissões automáticas é que as fronteiras de tempo entre os slots sejam corrompidas, atrasando ou até mesmo inibindo a transmissão do nodo dono do slot subsequente.

Para fins de prototipação, um transdutor externo 82C250 é necessário para conversão de valores digitais (RXD e TXD) em níveis de tensão diferenciais (CAN_H e CAN_L) que percorrem o barramento. O bloco PIC16F84A é um processador também descrito em linguagem VHDL que executa um conjunto de 35 instruções. A capacidade da memória de programa é de 1024 palavras de 14 bits cada enquanto que a memória de dados possui um espaço de endereçamento para 512 bytes. A interface do controlador CAN foi implementada de forma a poder conectá-lo diretamente no barramento de dados do processador. Logo, os registradores internos do controlador são visíveis na forma de endereços da memória de dados e podem ser acessados por meio de ponteiros em C. Na arquitetura básica o PIC16F84A exerce o papel do host, ou seja, é o processador responsável pela execução das tarefas da aplicação.

A segunda etapa marcou o início do desenvolvimento de uma extensão do protocolo CAN nativo visando o projeto de uma plataforma de comunicação para aplicações críticas de controle dinâmico. Basicamente, esta etapa consistiu no projeto de uma base de tempo global que permitisse o funcionamento de um controle de acesso tipo TDMA. Optou-se pela implementação do algoritmo de sincronização Daisy-Chain pela simplicidade e pelo fato de oferecer uma referência de tempo essencialmente distribuída de forma que não existe de fato nenhum mestre no sistema, ou seja, a base de tempo é mantida mesmo que um ou mais nodos quaisquer cessem subitamente de funcionar por motivo de falha.

Na última etapa de implementação foi desenvolvido em linguagem C um procedimento de criação inicial da base de tempo global. Conforme já mencionado, qualquer algoritmo de sincronização assume que inicialmente todos os relógios estão dentro dos limites de δ_{max} portanto a necessidade de uma primitiva da plataforma que tenha como função estabelecer a base de tempo quando o sistema é energizado pela primeira vez.

Uma vez finalizadas as etapas de implementação a plataforma foi validada em dispositivos FPGA de baixo custo onde o circuito do bloco lógico foi sintetizado com sucesso. Um sistema distribuído composto de 3 unidades foi prototipado em bancada de forma a verificar o correto funcionamento das primitivas de comunicação.

5.2 Organização Interna

A arquitetura proposta, resultante da segunda e terceira etapas do projeto é mostrada na Figura 5.2. Além dos blocos herdados da arquitetura básica foi implementado o bloco TDMA Clock que consiste na base de tempo local do nodo utilizada no controle de acesso e no procedimento de obtenção das leituras indiretas. O processador PIC passou a ser o núcleo da plataforma, responsável pelo gerenciamento de todo fluxo de dados entre a camada física e a camada da aplicação. Todos os demais blocos indicados na figura são visíveis como posições da memória de dados e são acessados com ajuda de ponteiros em linguagem C. O mapeamento de memória de CASCA é mostrado na tabela 5.1 onde são indicados somente as posições mais relevantes. No total são 20 registradores internos do processador mais outros 20 para controle, configuração e troca de dados com os blocos TDMA Clock e BSP.

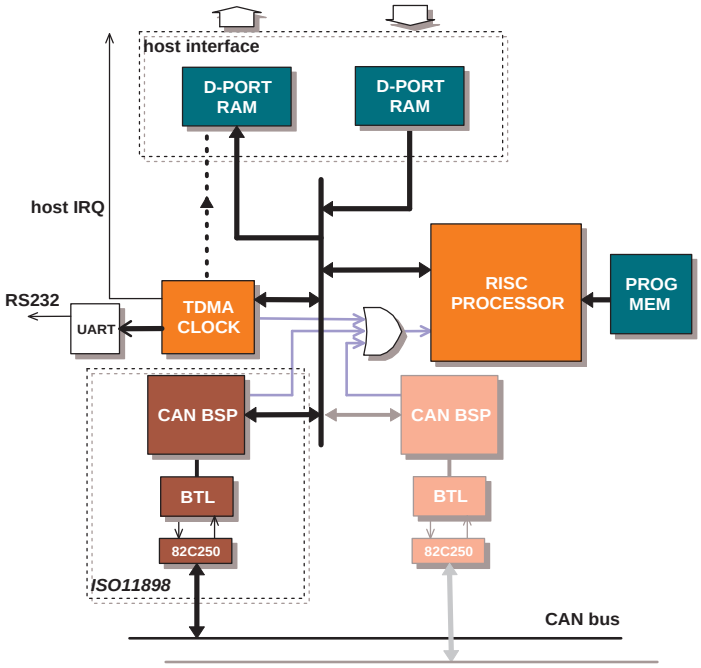

Figura 5.2: Organização interna da plataforma proposta.

Toda a comunicação entre blocos da plataforma ocorre através de um barramento interno paralelo de 8 bits de dados e 9 bits de endereço. Esta estrutura foi adotada para que fosse possível conectar todos os blocos ao barramento de dados do processador que no caso funciona como o árbitro num esquema tipo mestre-escravo. Os demais blocos não se comunicam diretamente entre si – toda e qualquer transferência de dados ocorre somente a partir de uma iniciativa do processador. Uma vantagem desta decisão de projeto é que qualquer um dos blocos que porventura falhem não conseguem interferir diretamente no funcionamento de outro sem antes confundir o núcleo. A arquitetura não conta atualmente com mecanismos de proteção em caso de falha no núcleo, que pode fazer por exemplo com que solicitações intermitentes de transmissão sejam enviadas ao BSP a todo momento. Como trabalhos futuros, dispositivos de guarda podem ser desenvolvidos para solucionar este problema.

CASCA foi desenvolvida visando oferecer suporte para o uso de técnicas de redundância ativa logo, o circuito pode ser sintetizado com canais de comunicação replicados. Para isto é necessário replicar o controlador CAN conforme indicado na tonalidade de cinza

claro. A conexão de outro bloco ao barramento interno tem impacto somente no código que roda no núcleo que deve considerar a existência do novo controlador no momento da transmissão e recepção de mensagens. Para o host a replicação é transparente.

O software que roda no processador PIC foi inteiramente desenvolvido em linguagem C. Para isto, foi utilizada a ferramenta SourceBoost (SOURCEBOOST IDE, 2005) a qual possui uma interface gráfica para auxiliar na escrita e depuração de código. O compilador gera um arquivo .hex que contém a seqüencia de instruções a ser carregada na memória de programa. Um script que recebe como entrada o arquivo .hex gerado cria uma descrição VHDL de memória ROM já com todo o código. A partir de então a plataforma pode ser simulada e sintetizada.

Tabela 5.1: Mapeamento da memória de dados do processador.

000	–	TMR0	PCL	STATUS	FSR	PORTA	TMR0	–
008	EEDATA	EEDADR	PCLATH	INTCON	–	–	–	–
010	–	–	–	–	–	–	–	–
–	–	–	–	–	–	–	–	–
080	–	OPTREG	PCL	STATUS	FSR	TRISA	TRISB	–
000	EECON1	EECON2	PCLATH	INTCON	–	–	–	–
–	–	–	–	–	–	–	–	–
1B8	–	–	–	CLKIRQSRC	MICROT0	MICROT1	MACROT0	MACROT1
1C0	MACROT2	NODE_ID	–	CLKIRQENA	CYCLE	SLOT	TXDATA0	TXDATA1
1C8	TXCTRL0	TXCTRL1	RXDATA0	RXDATA1	RXCTRL0	RXCTRL1	CANIRQSRC	CANIRQENA

5.2.1 Interface com o Host

O host se comunica com o controlador de rede na arquitetura CASCA através de 2 memórias dupla-porta conforme descrito a seguir:

`DPM-up`:

Memória dupla-porta de subida – host lê e controlador escreve. Utilizada para armazenar as mensagens que vão sendo recebidas pelo barramento.

`DPM-down`:

Memória dupla-porta de descida – host escreve e controlador lê. Utilizada para depositar as mensagens que serão transmitidas.

Todas as mensagens periódicas são definidas em tempo de projeto. Para cada mensagem consumida ou produzida pelo nodo existe um endereço associado na `DPM-down` e/ou `DPM-up`. Opcionalmente, filas de pequeno tamanho também podem ser alocadas para armazenamento temporário de mensagens assíncronas que não foram previstas. O processador troca informações com as DPMs com auxílio de um ponteiro chamado **ponteiro de mensagem** (`msgPtr`) que é atualizado em função do ciclo e slot corrente dentro do esquema TDMA.

O host também pode ler o relógio de CASCA a qualquer momento. Além disso, uma linha de interrupção proveniente do TDMA Clock vai até o host e pode ser utilizada para sincronizar a execução das tarefas no host com o esquema TDMA visando diminuir ao mínimo as latências de controle.

5.2.2 Sistema de Interrupções

As funções primitivas em software são implementadas na forma de rotinas de atendimento de interrupções disparadas pelos blocos TDMA Clock e BSP. Os sinais de interrupção de ambos (`tdma_irq` e `can_irq`) são concentrados no pino `PORTB(0)` – pino de interrupção externa do processador que recebe o resultado do OU lógico entre estes dois sinais. A origem da interrupção é então determinada a partir do resultado do OU lógico entre os valores dos registradores `CLKIRQSRC` e `CANIRQSRC`. Todas as possíveis fontes são descritas a seguir:

`MyAtIrq=0x40`

Início do slot de transmissão dentro de um ciclo TDMA. Disparada pelo TDMA Clock sempre que o contador de slots coincide com o endereço do nodo.

`RoundIrq=0x20`

Interrupção de temporização disparada pelo TDMA Clock após cMacrotickMax unidades de tempo desde a última requisição onde cMacrotickMax é o tempo de duração total de um ciclo TDMA. A requisição por sua vez é feita habilitando a respectiva interrupção.

`SofIrq=0x10`

Início de um novo quadro. Esta interrupção é disparada sempre que o controlador CAN detecta uma transição recessivo→dominante no barramento indicando o início da transmissão de um quadro no barramento.

`ErrIrq=0x08`

Interrupção gerada sempre o controlador CAN detecta algum erro em nível de enlace. Maiores informações sobre detecção de erros no protocolo CAN são encontradas em (CIA, 1999).

`TxDIrq=0x04`

Transmissão de um quadro bem sucedida, gerada pelo BSP se após a completa transmissão do quadro nenhum erro tiver sido detectado.

`RxDIrq=0x02`

Recepção de um quadro bem sucedida, gerada pelo BSP se após a completa recepção de um quadro com CRC correto.

`CollIrq=0x01`

Colisão de acesso detectada. Interrupção gerada quando o controlador CAN perde o acesso durante a fase de arbitração.

Interrupções externas devem ser ativadas setando o bit `INTCON(4)`. A cada pulso no pino `PORTB(0)` o contador de programa (`PC`) do núcleo é imediatamente carregado com o endereço `0004h` a partir de onde a rotina de interrupção deve ser programada.

5.3 Funcionamento do Protocolo

O funcionamento do protocolo da arquitetura CASCA se baseia numa hierarquia de tempo que define em cada nível a granularidade de um componente referente ao controle da base de tempo global. Esta idéia foi herdada do protocolo FlexRay por proporcionar mais clareza no entendimento, projeto e uso do sistema como um todo.

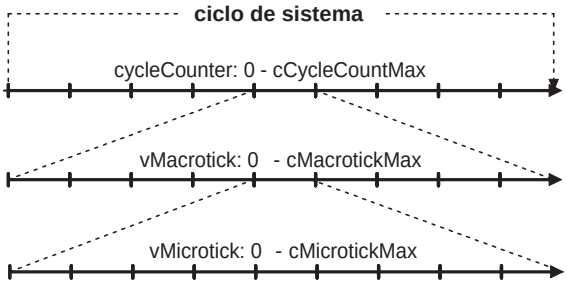

Figura 5.3: Hierarquia de tempo.

Na Figura 5.3 o nível mais abaixo representa o nível de microtick que consiste na fonte primária de tempo do sistema. O tamanho de 1 microtick deve ser igual ao período do sinal gerado pelo dispositivo oscilador da plataforma, ou seja, um período de clock. Em situações particulares, quando o oscilador é muito rápido o microtick pode ser opcionalmente gerado por um divisor de freqüência.

Logo acima está o nível de macrotick que consiste na unidade de mínima de representação da base de tempo global. Por isso, o tamanho de 1 macrotick deve ser maior ou igual a precisão δ_{max} que o algoritmo de sincronização consegue garantir. No nível mais alto da hierarquia está representado o ciclo de sistema implementado na forma de um contador de ciclos (ou rodadas) TDMA.

O particionamento dos ciclos TDMA em slots de tempo é feito no nível de macrotick, i.e. cada slot tem seu instante de início bem definido. Visto que todos os slots são de mesmo tamanho, a relação entre o tamanho de um ciclo TDMA (`cMacrotickMax`) e o número de slots por ciclo TDMA (`cNumberOfNodes`) deve resultar em um valor inteiro de macroticks.

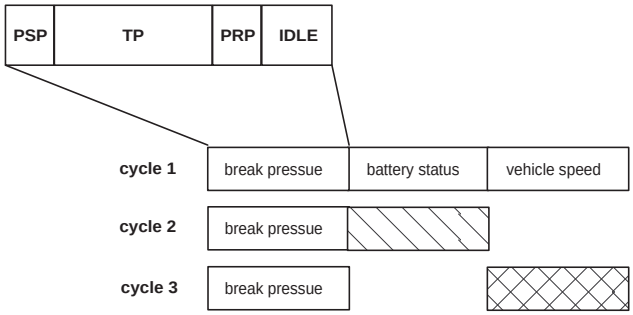

Figura 5.4: Exemplo de um ciclo de sistema com 3 ciclos TDMA.

A Figura 5.4 mostra um exemplo de ciclo de sistema formado por apenas 3 ciclos TDMA com 3 slots de tempo cada um – indicando que no sistema existem somente 3 nodos. O tamanho do slot deve ser grande o suficiente para cobrir o tempo de transmissão da mensagem (TP) além da fase de pré-transmissão (PSP) e de pós-recepção (PRP) cujo

tempo de processamento foi extraído por simulação. A fase IDLE representa o restante de tempo onde o núcleo está em laço infinito aguardando nova interrupção.

A fase de PSP representa a latência da interrupção `MySlot` mais o tempo de busca da mensagem nas memórias de interface. Neste momento `msgPtr` já aponta para a posição correta na `DPM-down` bastando carregar o seu conteúdo para os registradores de transmissão do controlador CAN. Mesmo que o host de um nodo transmissor não consiga produzir dados a tempo o quadro é mesmo assim enviado. Isto porque o algoritmo de sincronização precisa realizar leituras com muita freqüência caso contrário a base de tempo global pode divergir além dos limites de δ_{max}. Para que o(s) receptor(es) consigam diferenciar valores válidos e inválidos foi utilizado o bit `r0` do quadro CAN como uma flag indicativa.

Durante a fase PRP o núcleo faz o processamento da informação contida no quadro recebido. Caso o barramento tenha sido replicado as mensagens redundantes são comparadas antes de atualizar a `DPM-up` respeitando um critério predefinido. Além disso, a cada quadro recebido o processador verifica a consistência temporal comparando o número do ciclo TDMA e do slot corrente fornecidos pelo transmissor no corpo da mensagem com os respectivos valores dos contadores locais. Caso a comparação falhe, então o processador assume que está "perdido" – fora de sincronia com os demais – e reinicia o procedimento de inicialização descrito na seção 5.4.

O tempo de transmissão do quadro é dado pela seguinte relação:

$$TP = \frac{M + s}{v} \tag{5.1}$$

onde v é a taxa de transmissão no barramento CAN em bps (bits por segundo), M o tamanho do quadro CAN dado em número de bits e s o número de bits redundantes que o protocolo CAN automaticamente insere durante a transmissão do quadro no objetivo de forçar a ocorrência de transições $1 \rightarrow 0$ necessárias para o reajuste do tempo de bit.

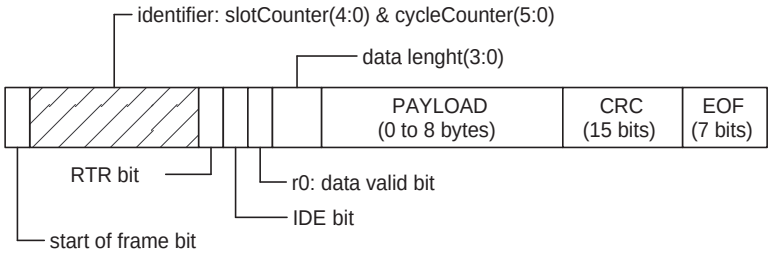

Figura 5.5: Quadro CAN no formato padrão.

Na Figura 5.5 é mostrado o formato de um quadro. No total são 19 bits de controle no cabeçalho, 0 a 8 bytes no campo de dados, CRC de 15 bits e um delimitador de fim de quadro (EOF) com 7 bits. Considerando que o campo de dados contém 2 bytes tem-se que M=57. No pior caso, a cada 5 bits de mesma polaridade o BSP insere 1 bit redundante logo tem-se que s=M/5. Logo, o tempo de transmissão do quadro sobre um barramento CAN com taxa de 1Mpbs é igual a:

$$TP = \frac{57 + \frac{57}{5}}{10^6} \simeq 69\mu s \tag{5.2}$$

Observa-se que todo o quadro carrega informação explícita de tempo para verificação da consistência temporal. O valor do contador de slots (`slotCounter`) é colocado

nos 5 bits mais significativos do identificador enquanto que o número do ciclo corrente (`cycleCounter`) é colocado nos 6 bits menos significativos. Isto garante que em caso de colisão durante o processo de inicialização o nodo com endereço menor sempre ganha o barramento.

5.4 Gerenciamento do Tempo

Da mesma forma que nos demais protocolos que adotam o modelo time-triggered, na arquitetura CASCA o tempo é definido dentro de um horizonte finito chamado de **ciclo de sistema** sobre o qual as ações de natureza periódica são previamente agendadas. O ciclo de sistema é por sua vez subdividido em **rodadas TDMA**, e as rodadas TDMA por sua vez subdivididas em slots de tempo de **mesmo tamanho**. Cada nodo possui um slot dedicado dentro de cada rodada TDMA para transmitir suas mensagens. A subdivisão do ciclo de sistema em rodadas distintas permite que diferentes seqüências de mensagens sejam transmitidas. Um mesmo nodo pode transmitir 2 ou mais mensagens com diferentes periodicidades.

Em decorrência do tamanho dos contadores do módulo TDMA Clock, o maior ciclo de sistema possível de ser programado possui $2^6 = 64$ rodadas TDMA com no máximo $2^5 = 32$ slots cada um.

A base de tempo global, formada por todos os relógios sujeitos a correções sucessivas, progride ao longo do ciclo de sistema e interrupções provenientes do TDMA Clock disparam as transmissões a cada início de um novo slot caracterizando um esquema de multiplexagem no tempo.

Sob uma perspectiva macro a base de tempo global é representada da seguinte forma:

:`cycleCounter`:`vMacrotick`:

No total somente 3 posições de memória são necessárias: 1 byte para o contador de ciclo e 2 bytes para o contador de macroticks. Relações de tempo e causalidade podem ser controladas na camada de aplicação programando interrupções ou simplesmente consultando o tempo de ciclo quando necessário. Garante-se assim que réplicas funcionais podem trabalhar em perfeita sincronia sem a necessidade de mecanismos de sincronização em mais alto nível.

5.4.1 Sincronização de Relógios

Um sistema distribuído construído sobre a plataforma CASCA possui uma base de tempo global mantida pela execução do algoritmo Daisy-Chain (DCA), conforme apresentado em (CARVALHO et al., 2006a) e (CARVALHO et al., 2006b). Trata-se de um método de sincronização interna onde um novo fator de correção é calculado e aplicado localmente a cada novo quadro transmitido no barramento. Ao contrário dos demais métodos apresentados no capítulo 3, o princípio de funcionamento do DCA se baseia no cálculo do fator de correção a partir de apenas uma leitura. A idéia é fazer com quem todos os nodos ajustem seus relógios em função de uma leitura indireta da base de tempo do nodo transmissor da vez.

A principal vantagem do DCA está no fato de não ser necessário armazenar temporariamente as leituras uma vez que para cada novo desvio calculado sua utilização é imediata, podendo ser descartado logo em seguida. Logo, a implementação híbrida do DCA representa uma solução mais econômica em comparação à outros algoritmos tais

como o FTM, FTA e CNV pois seu circuito/código é bem mais simples. O fato de ser mais simples implica diretamente que falhas são menos prováveis. Além disso, não existe a necessidade de agendar instantes de execução do algoritmo uma vez que, por definição, cada nodo deve calcular o fator de correção e aplicá-lo imediatamente a cada leitura indireta realizada no controlador. No protocolo TTP/C, por exemplo, é preciso configurar em tempo de projeto os slots onde a sincronização ocorre.

O intervalo de sincronização R_{int} varia conforme a distância entre duas transmissões sucessivas no barramento e depende fortemente do agendamento de transmissões ao longo do ciclo de sistema. O menor valor de R_{int} acontece no momento de duas transmissões consecutivas onde R_{int} é igual ao tamanho de 1 slot. Sabe-se no entanto que o ciclo de sistema pode conter alguns slots vazios onde nenhum ajuste de relógios é feito. Neste caso o intervalo entre duas correções sucessivas pode se estender para até $(\varphi + 1)\tau_{slot}$ onde φ é o número de slots vazios consecutivos e τ_{slot} é o tamanho de um slot de tempo, dado em unidades de segundo (μs, ms, etc). Além disso, existe também a possibilidade de um nodo falhar e não enviar nenhuma mensagem no slot de tempo esperado. Do ponto de vista da sincronização, esta falha têm o mesmo efeito de um slot vazio na medida em que nenhuma nova correção é feita.

Apesar do efeito de slots vazios sobre R_{int}, no algoritmo Daisy-Chain as correções são feitas com maior freqüência. Isto significa dizer que o efeito de ρ sobre a precisão do sistema é minimizado, lembrando sempre que δ_{max} é diretamente proporcional à ρR_{int}. Com isso, é possível que osciladores menos precisos sejam utilizados como fontes primárias de tempo diminuindo assim o custo total do projeto. Considerando também que diferenças bruscas de temperatura aumentam ainda mais a variação na freqüência, pode-se afirmar que o uso do algoritmo Daisy-Chain resulta em um sistema mais robusto no que se refere à falhas causadas por agentes externos.

5.4.2 Mecanismo de Leitura Indireta

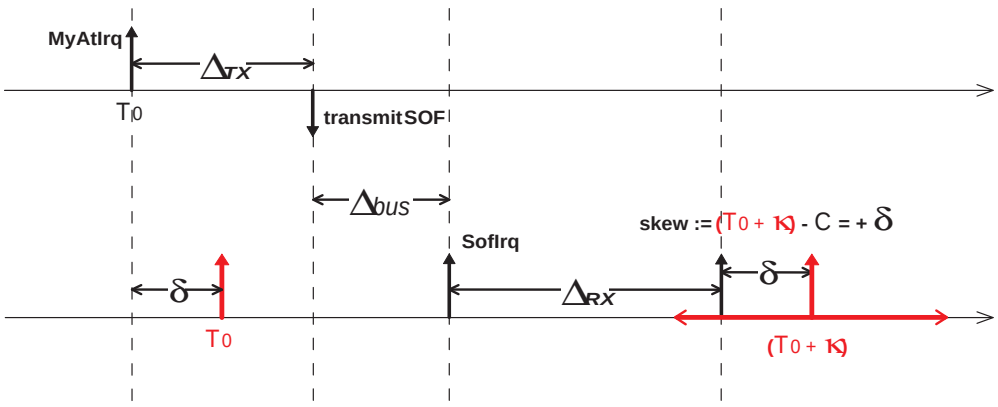

Figura 5.6: Método de leitura indireta.

O mecanismo de leitura do relógio do transmissor da vez é mostrado na Figura 5.6. No momento em que a base de tempo do transmissor atinge o seu instante de transmissão, uma interrupção (`MyAtIrq`) é gerada pelo módulo TDMA Clock dando início ao tratamento da respectiva rotina no núcleo buscando na `DPM-down` a mensagem a ser enviada no slot corrente. Na Figura, Δ_{TX} representa a latência de saída resultante em sua maior parte da rotina de tratamento de interrupção no núcleo.

No lado receptor, quando o bit de início do quadro é detectado o módulo BSP dispara

uma interrupção (`SofIrq`) marcando o instante em que a leitura indireta deve ser efetuada. Após o tratamento da interrupção no núcleo, a diferença entre o sua base de tempo do receptor e a base do transmissor é inferida a partir da diferença entre o instante esperado e o valor do relógio local no momento da leitura. A latência de saída Δ_{RX} também é resultado da execução da rotina de tratamento de interrupção.

Para compensar o efeito do atraso de propagação o instante esperado é definido como sendo o instante de transmissão (do ponto de vista do receptor) somado com o fator de compensação κ. As componentes Δ_{TX} e Δ_{RX} do fator de compensação foram obtidas por simulação enquanto que a componente Δ_{bus} foi extraída por meio de protótipo.

No DCA, o desvio inferido é aplicado imediatamente após a obtenção da leitura. Com isto, elimina-se em grande parte do problema conhecido como envelhecimento das leituras. Nos protocolos TTP/C e FlexRay, por exemplo, as leituras são feitas em série ao longo de um intervalo R_{int}. No momento em que o fator de correção é aplicado as leituras coletadas já estão "velhas", isto é, o erro associado é maior devido ao efeito de ρ durante R_{int}. Neste caso, a primeira leitura feita no início de R_{int} será obviamente a mais prejudicada. Por outro lado, no DCA o efeito do envelhecimento pode ser considerado desprezível.

Para fins de proteção contra leituras incorretas resultantes de nodos que porventura transmitam fora do instante esperado, a versão tolerante à falhas do DCA propõe a implementação de uma janela de recepção cujo centro coincide com o instante esperado. Neste caso, são consideradas incorretas e rejeitadas todas as leituras que fornecem como resultado um desvio que excede os limites da janela de recepção. Para evitar que leituras corretas sejam descartadas o tamanho desta janela deve ser $\geq 2(\delta_{max} + \varepsilon)$. O DCA é capaz de tolerar falhas simétricas e bizantinas que porventura venham a afetar o mecanismo de leitura. A condição essencial para que a base de tempo se mantenha dentro dos limites de δ_{max} é que leituras defeituosas ocorram com menos freqüência que leituras corretas. As relações entre número de falhas e a precisão máxima do sistema são apresentadas em (LONN, 1999b).

5.4.3 Correção da Base de Tempo Local

O desvio, e.g. o fator de correção, é aplicado na base de tempo local mesmo que seja detectado um erro no CRC do quadro recebido pois assume-se que mensagens inconsistentes não prejudicam o algoritmo quando transmitidas nos instantes corretos.

Considerando que o início de um quadro CAN consiste numa transição de $1 \rightarrow 0$ no nível lógico do barramento, glitches podem fazer com que o BSP dispare uma interrupção `SofIrq` gerando como resultado uma leitura falsa. Para contornar este problema, a versão tolerante à falhas do algoritmo sugere também que apenas uma fração do fator de correção seja aplicado. A função de convergência do DCA se resume ao seguinte cálculo:

$$\frac{\delta_{i,j}}{r} \tag{5.3}$$

onde r é um inteiro diferente de zero que deve ser estabelecido a priori. Assim, garante-se que o efeito de uma leitura falsa seja amenizado.

O DCA foi implementado na forma de uma combinação de hardware e software. As interrupções são geradas pelos blocos TDMA Clock e BSP que por sua vez disparam rotinas de tratamento no núcleo. Estas rotinas se resumem na decodificação da origem da interrupção e no envio de comandos ao bloco TDMA Clock para cálculo do desvio e correção do relógio local conforme descrito a seguir:

`GET_SKEW`:

Infere o desvio da base de tempo local em relação à base de tempo do transmissor.

`APPLY_CORR_TERM`:

Ordem para calcular o fator de correção conforme a equação 5.3 e aplicá-lo na base de tempo local. Por simplicidade o cálculo é feito na forma de deslocamento binário logo r é uma potência, ou seja:

$$r=2^l$$

onde l é o número de bits de $\delta_{i,j}$ deslocados para a direita.

`RESET`:

Reinicializa todos os contadores que formam a base de tempo.

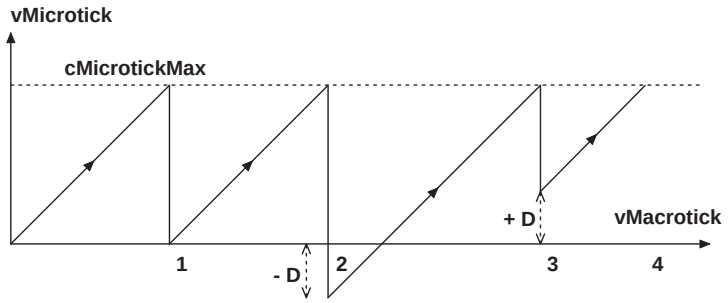

Figura 5.7: Comportamento do contador de microtick.

O fator de correção é aplicado de uma vez só caracterizando um método de correção discreta, conforme mostrado na Figura 5.7. Quando o núcleo envia o comando `APPLY_CORR_TERM` para o bloco TDMA Clock uma requisição fica pendente até o momento do próximo estouro do contador de microtick – instante em que a base de tempo local é efetivamente modificada (retrocede ou avança). O valor $\pm$**D** representa o fator de correção calculado a partir da última leitura fazendo o tamanho de 1 macrotick contrair ou dilatar para que a base de tempo se ajuste à base do transmissor da vez.

Observa-se que o efeito das correções é uma pequena variação do tamanho do macrotick ao longo do tempo. Contudo, o impacto deste efeito na aplicação é mínimo pois a amplitude máxima desta variação – igual à metade da largura do macrotick – representa uma parcela muito pequena do tamanho de 1 ciclo TDMA.

Uma vez que, por definição, o módulo do fator de correção nunca excede o tamanho de 1 macrotick, isto é:

$$|\ \mathbf{D}\ | < \texttt{cMicrotickMax}$$

garante-se que a base de tempo global de CASCA é monotônica uma vez que nenhum valor de macrotick desaparece nem acontece duas vezes no mesmo ciclo de sistema.

5.4.4 Criação da Base de Tempo

Quando o sistema é energizado pela primeira vez não existe base de tempo global pois não há nenhum mecanismo que garanta que os relógios dos nodos envolvidos sejam iniciados exatamente ao mesmo tempo. Logo, o objetivo da primitiva de criação da base de tempo é trazer todos os relógios para dentro dos limites de δ_{max} para que então o algoritmo de sincronização e o esquema TDMA possam funcionar.

A plataforma proposta possui um método próprio que se beneficia do mecanismo de arbitração não-destrutivo do protocolo CAN nativo para resolver colisões iniciais que porventura aconteçam. Deste modo, garante-se que sincronização interna seja atingida rapidamente sem a necessidade de um controle centralizado. Qualquer nodo pode criar a base de tempo e iniciar o esquema TDMA.

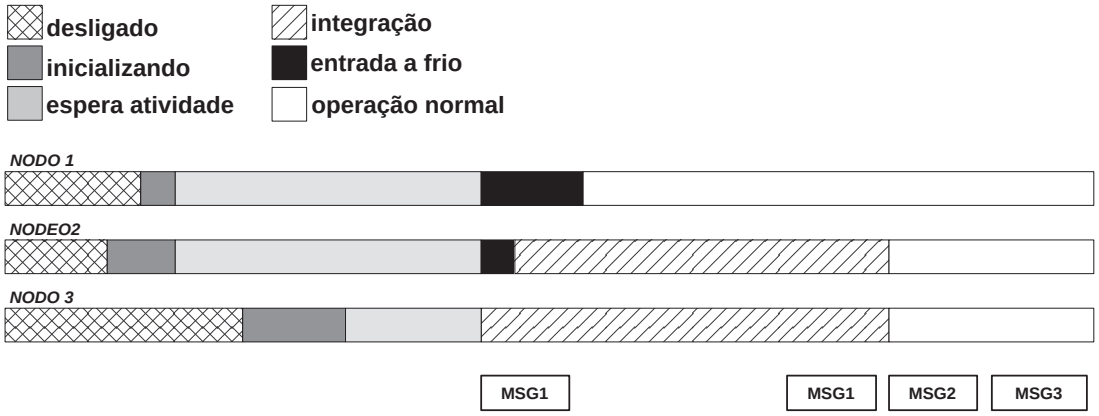

Figura 5.8: Procedimento de entrada a frio.

A Figura 5.8 mostra o exemplo simplificado de uma situação onde colisões de acesso podem ocorrer. Supondo que inicialmente 3 nodos são energizados em instante de tempo muito próximos cada um transiciona do estado **desligado** para **inicializando** onde o núcleo inicializa alguns registradores de configuração do próprio processador e dos demais blocos da plataforma e faz a carga do seu endereço disponível em 5 pinos de entrada. Em seguida os nodos entram na fase **espera atividade** durante a qual o núcleo aguarda por um período de tempo igual à 2 ciclos TDMA no intuito de verificar se já existe atividade de comunicação em curso. Se ao final da fase **espera atividade** nenhuma interrupção `SofIrq` ter sido disparada então o núcleo assume que está sozinho no barramento e inicia o procedimento de criação da base de tempo que recebe o nome de entrada a frio, representado pela fase **entrada a frio**. No exemplo mostrado, os nodos 1 e 2 iniciam uma entrada a frio ao mesmo tempo e como resultado seus quadros colidem. Neste caso o nodo 1 transmite um identificador de maior prioridade ganha o barramento e transmite seu quadro sem mesmo perceber que a colisão ocorreu.

A partir do momento em que o primeiro quadro é transmitido com sucesso o nodo transmissor recebe o atributo de **líder** do ponto de vista dos demais. O líder ao receber a confirmação do BSP que a transmissão foi bem sucedida transiciona diretamente para o estado **operação normal** e a partir de então o núcleo começa a acessar as DPMs. Em relação aos demais nodos, 2 situações são possíveis:

1. No caso de colisão inicial entre 1 ou mais nodos, todos os nodos que perderam o acesso ao barramento pegam a informação explícita de tempo contida no identifi-

cador do quadro transmitido pelo líder e modifica os contadores de slot e de ciclo TDMA no TDMA Clock com este valor.

Nos níveis de macrotick e microtick nada precisa ser feito pois o fato de ter ocorrido colisão indica que estes contadores já estão alinhados dentro dos limites de δ_{max}.

2. Todos os demais nodos detectam o início da transmissão de algum quadro durante a fase espera atividade. Quando o BSP dispara a interrupção `SofIrq` o núcleo carrega os contadores de microtick e macrotick com o valor do instante esperado que consiste no tamanho do atraso de propagação e entra imediatamente na fase **integração** representando uma intenção de adotar a base de tempo global já existente. Após receber o primeiro quadro por completo, o receptor integrante pega a informação explícita de tempo contida no identificador e modifica os contadores de slot e de ciclo TDMA no TDMA Clock com este valor.

   A partir de então aguarda-se um segundo quadro proveniente do mesmo nodo por um intervalo de tempo igual à 1 ciclo TDMA. Após a recepção deste segundo quadro o núcleo verifica a consistência temporal do transmissor e entra no estado **operação normal** caso a comparação entre os contadores de slot e ciclo TDMA tenha sido bem sucedida.

Uma vez estabelecida a base de tempo inicial o algoritmo Daisy-Chain entra em ação para manter todos os relógios suficiente próximos uns dos outros. Vale salientar mais uma vez que ambas as funções primitivas de sincronização de relógios e criação da base de tempo são essencialmente distribuídas uma vez que sua execução não depende de nenhuma unidade específica.

Ambas as primitivas de sincronização e criação da base de tempo ficam confinadas dentro da plataforma e são executadas de forma transparente para a camada da aplicação.

5.5 Metodologia de Projeto

Em vista do comportamento estático do esquema TDMA a plataforma CASCA deve ser configurada em tempo de projeto visando atender os requisitos de comunicação de um sistema específico. Dado um conjunto de requisitos na forma <P_i,L_i> onde P_i é o período e L_i a largura da mensagem i dado em bits, cabe ao projetista definir um ciclo de sistema que consiga alocar tempo de transmissão dedicado para todas as mensagens de forma a assegurar o particionamento temporal.

O primeiro passo é tomar conhecimento de qual o período mais “apertado”, ou seja, a dinâmica mais rápida imposta pelo objeto ou processo controlado para que se possa normalizar os requisitos e construir o ciclo de sistema. Assumindo que o menor período seja P_{min} todos os períodos podem ser expressos na forma de valores inteiros múltiplos de P_{min}, inclusive P_{min} que passa a ser 1. Por exemplo, se em um dado sistema os requisitos de comunicação forem <1.5ms,16>, <3ms,1> e <7.5ms,8> então eles passam a ser expressos como <1,16>, <2,1> e <5,8> onde P_{min} é igual a 1.5ms.

Requisitos com períodos não-múltiplos de P_{min} não devem existir. Caso isto ocorra o fator de normalização pode ser uma fração de P_{min}, e.g. $\frac{1}{2}P_{min}$, $\frac{1}{3}P_{min}$, $\frac{2}{3}P_{min}$ e assim por diante. Supondo agora que em uma segunda aplicação os requisitos sejam <2.5ms,16>, <3ms,1> e <4ms,8>, neste caso seria necessário normalizá-los em função de 0.5ms resultando em <5,16>, <6,1> e <8,8>.

Tomando como base os requisitos normalizados, a definição do ciclo de sistema consiste na escolha dos seguintes parâmetros:

`cNumerOfNodes`:

Quantidade de nodos conectados ao barramento. Determina o número de slots que irá existir dentro de cada ciclo TDMA.

`cMicrotickMax`:

Referente ao tamanho de 1 macrotick que representa a granularidade mínima da base de tempo global. A escolha deste valor recai sobre uma estimativa da precisão δ_{max} da base de tempo que o DCA consegue garantir. Logo, este parâmetro é diretamente influenciado por ρ, R_{int}, e pela freqüência f_{ck} da fonte primária de tempo da plataforma. Assumindo que o tamanho de 1 microtick é igual a 1 período de clock então o tamanho de 1 macrotick é definido como:

$$\texttt{cMicrotickMax} = \delta_{max}.f_{ck}$$

onde:

$$\delta_{max} = 2\varepsilon + \rho R_{int} + \frac{4}{f_{ck}}$$

O último termo da equação de ε representa a variação da latência interna do mecanismo de leitura indireta, percebido no momento da simulação.

`cMacrotickMax`:

Referente ao tamanho de 1 ciclo TDMA. Para que se possa mapear os requisitos tem-se que:

$$\texttt{cMacrotickMax.cMicrotickMax}.f_{ck}^{-1} = \phi$$

onde ϕ é o fator de normalização. Além disso, o valor deste parâmetro deve ser um múltiplo de `cNumerOfNodes` considerando que todos os slots de tempo são de mesmo tamanho.

`cCycleCountMax`:

Tamanho do ciclo de sistema expresso em número de ciclos TDMA. Este parâmetro é função direta do mínimo múltiplo comum (MMC) de todos os períodos envolvidos no conjunto de requisitos do sistema. O tamanho do ciclo de sistema pode extrapolar caso muitos dos períodos normalizados sejam inteiros primos conforme ilustra a Tabela 5.3.

Além destes 4 parâmetros também é preciso definir em tempo de projeto as posições nas memórias `DPM-up` e `DPM-down` nas quais as mensagens são trocadas entre o host e a pilha de rede.

Em regime normal de operação as tarefas da aplicação se comunicam conforme um modelo do tipo produtor-consumidor onde o consumidor conhece o instante em que o produtor transmite a mensagem.

Tabela 5.3: Tamanho do ciclo de sistema em função dos períodos das mensagens.

	P1	*P2*	*P3*	`cCycleCountMax`
a)	*1*	*2*	*2*	*2*
b)	*1*	*3*	*5*	*15*
c)	*1*	*3*	*6*	*6*
d)	*1*	*3*	*4*	*12*
e)	*1*	*3*	*7*	*21*
f)	***2***	***3***	***7***	***42***
g)	*2*	*4*	*7*	*28*
h)	*2*	*3*	*8*	*24*

5.5.1 Alocação de Tráfego Assíncrono

A plataforma proposta se apóia na idéia de que o tráfego cíclico deve ser priorizado. No entanto, sabe-se que solicitações aperiódicas de transmissão também são importantes para o mapeamento de alarmes e tarefas de configuração e diagnóstico. Para isto propõem-se duas soluções:

1. Utilizar os slots vazios (buracos) resultantes da definição do ciclo de sistema.

   A partir do exemplo mostrado na Figura 5.4 verifica-se que nem todos os slots são utilizados pois na prática os períodos de transmissão das mensagens são diferentes. Sendo assim, é possível que mensagens de natureza assíncrona podem ser dinamicamente alocadas em um destes buracos no ciclo de sistema. Caso 2 ou mais nodos tentem transmitir ao mesmo tempo o mecanismo de resolução não-destrutivo do protocolo CAN original entra em ação fazendo com que o nodo com endereço (identificador) menor ganhe o acesso.

2. Assumir uma periodicidade mínima e alocar um slot dedicado.

   Neste caso a mensagem de natureza assíncrona deve ser considerada na fase de projeto como um requisito de comunicação periódico. Em função da taxa média de transmissões previstas em regime normal de operação assume-se um período aproximado para que um slot exclusivo seja alocado para esta mensagem dentro do ciclo de sistema.

5.5.2 Modelo de Falhas

No que se refere ao projeto de um sistema crítico parte-se do pressuposto que o bloco lógico de CASCA é livre de falhas de projeto e que a tecnologia de fabricação utilizada na sua síntese é suficientemente robusta para que falhas estruturais do circuito (curtos e abertos) possam ser desconsideradas. Logo, técnicas de tolerância a falhas focam no mascaramento de falhas no host e na infraestrutura física da rede. A plataforma CASCA oferece suporte para a aplicação de técnicas de diversidade de versões, redundância ativa e unidades de reserva pois existe uma base de tempo global que permite que unidades replicadas trabalhem em perfeita sincronia.

Um sistema distribuído construído sobre CASCA é capaz de mascarar, de forma transparente para a camada da aplicação, qualquer uma das falhas do seguinte conjunto:

Manifestas:

1. Transientes (glitches) no barramento.

   Assume-se que o número de bits do quadro corrompidos por glitches é menor que a distância de hamming do polinômio CRC gerador utilizado no protocolo CAN. Assim, garante-se que falhas causadas por glitches sejam detectadas pela implementação da camada de enlace.

2. Falha no barramento.

   Ruptura de `CAN_H` ou `CAN_L`, curto de `CAN_H` ou `CAN_L` com `GND` ou `VCC`.

3. Destruição do nodo.

   Qualquer falha que interrompa toda e qualquer atividade no nodo como por exemplo perda do contato com `VCC` ou `GND`, destruição da fonte primária de tempo, entre outros.

Simétricas:

1. Ruído (homogêneo) no barramento

   Caso um número de bits maior que a distância de hamming seja corrompido o checksum pode não detectar o erro.

2. Falha de I/O

   Qualquer falha nos sensores, conversores A/D, etc de interface entre o host e o objeto ou processo controlado.

3. Inversão de bits de memória no host.

4. Falha de projeto no host.

   Erro de resultados calculados no host.

Bizantinas:

1. Ruído (heterogêneo) no barramento.

   Considera-se este o único caso onde uma falha bizantina pode ocorrer. Na prática é possível que o erro causado pelo ruído seja diferente em função da posição dos nodos. Logo, uma mesma mensagem enviada pode ser recebida com valores diferentes em 2 ou mais nodos.

Conforme a falha varia a técnica adotada e a atitude tomada pelos receptores ao receber mensagens redundantes. No caso de falhas simétricas ou bizantinas mensagens redundantes são sempre comparadas ao passo que para se proteger contra falhas manifestas basta utilizar qualquer uma desde que a mensagem não tenha erro de CRC.

Observa-se que, apesar de conter praticamente todas as falhas que ocorrem na prática, o modelo proposto é relativamente restrito quando comparado a modelos teóricos mais completos (AVIZIENIS, 2004). Por exemplo, falhas de natureza humana em tempo de execução causadas por mau-uso não são consideradas pois assume-se que interação com o usuário final é mínima ou inexistente. Caso o sistema esteja sujeito a outras falhas não presentes no modelo será então necessário recorrer a métodos de projeto e/ou mecanismos de proteção para diminuir a respectiva taxa de ocorrência.

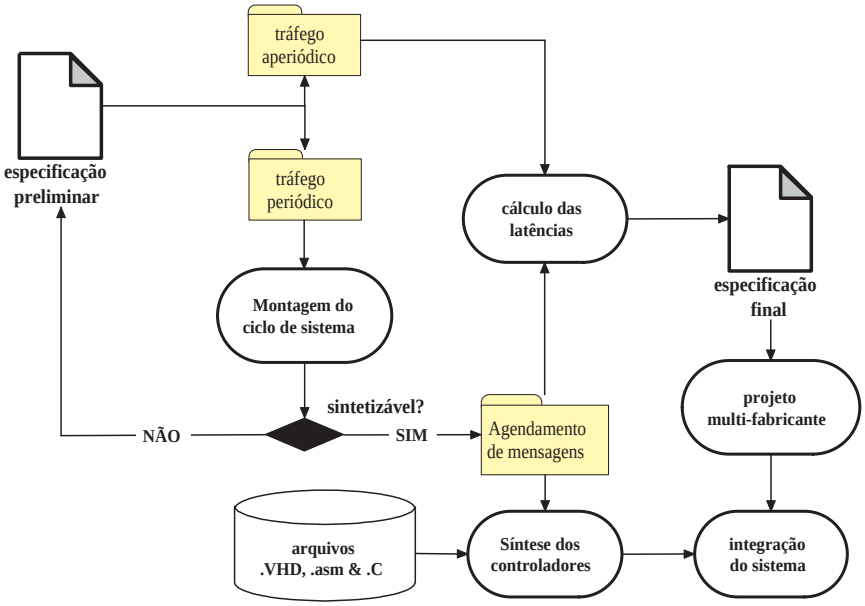

Figura 5.9: Fluxo de projeto proposto.

5.5.3 Fluxo de Projeto Proposto

CASCA garante particionamento temporal logo, subsistemas (independentes ou não) podem ser desenvolvidos por equipes distintas para que posteriormente sejam integrados sobre um mesmo barramento sem nenhum efeito secundário nos tempos de resposta. Na Figura 5.9 é mostrado o fluxo de projeto proposto. Parte-se de uma especificação preliminar onde estão todos os requisitos de comunicação impostos ao sistema. Estes requisitos por sua vez são separados em função da sua regularidade, podendo ser diferenciados em trafego periódico ou aperiódico. Requisitos periódicos são utilizados como entrada para o processo de definição do ciclo de sistema enquanto que requisitos aperiódicos (ou dinâmicos) são alocados após a definição do mesmo. Em função do nível de confiabilidade exigido na aplicação mensagens redundantes podem ser previstas na especificação final.

A condição essencial para que o ciclo de sistema seja sintetizável é que todos os períodos envolvidos sejam múltiplos do menor para que a normalização possa ser feita conforme já discutido. Caso esta condição não seja satisfeita os períodos não múltiplos devem ser reajustados (diminuídos ou aumentados). Uma vez normalizados é possível dar início à inserção dos requisitos de comunicação dentro de um horizonte de tempo finito. Eventuais modificações poderão feitas caso o tamanho do ciclo de sistema extrapole em decorrência da existência de muitos períodos normalizados com valores primos. É importante salientar que a modificação deve ser sempre na direção do múltiplo menor para que a performance do sistema de controle não seja comprometida.

Uma vez definido o ciclo de sistema, e.g. o comportamento do esquema TDMA, podem ser calculadas as latências máximas impostas ao tráfego assíncrono, que devem ser por sua vez reportadas na documentação final do projeto. A plataforma é sintetizada com os fontes da biblioteca (.vhd, .c & .asm) informando todos os parâmetros obrigatórios mencionados anteriormente. Todos os nodos do sistema devem ser construídos com a mesma configuração no bloco lógico ao passo que os hosts podem ser inteiramente desenvolvidos or equipes distintas como subsistemas independentes para depois serem integrados sem nenhum efeito secundário nos tempos de resposta previstos.

5.6 Estudo de Caso

battery	vehicle controller	inverter	transmission control	driver inputs	brakes
traction battery ground fault (1 bit)	clutch pressure control & torque command (16 bits)	*Main Contactor Close*	transaction clutch line pressure (8 bits)	*Key Switch Run*	cylinder and line break pressure (16 bits)
high and low contactor control (16 bits)	*Reverse and 2nd Gear Clutches*	torque measured & processed motor speed (16 bits)	transaxle lubrication pressure (8 bits)	accelerator position (8 bits)	vehicle speed (8 bits)
DC/DC converter (1 bit)	clutch pressure control & torque command (16 bits)	*FWD/REV Ack.*	transaction clutch line pressure (8 bits)	*Key Switch Start*	cylinder and line break pressure (16 bits)
traction battery voltage & current (16 bits)	*Brake Solenoid*	torque measured & processed motor speed (16 bits)	*Motor/Trans Over Temperature*	accelerator position (8 bits)	*Brake Switch*
traction battery ground fault test (2 bits)	clutch pressure control & torque command (16 bits)	*Inhibit*	transaction clutch line pressure (8 bits)	*Accelerator Switch*	cylinder and line break pressure (16 bits)
high and low contactor control (16 bits)	*Backup Alarm*	torque measured & processed motor speed (16 bits)		accelerator position (8 bits)	
Hi&Lo Contactor Open/Close	clutch pressure control & torque command (16 bits)	*Status/Malfunction (TBD)*	transaction clutch line pressure (8 bits)	*Emergency Brake*	cylinder and line break pressure (16 bits)
auxiliar battery voltage & current (16 bits)	*Warning Lights (7 bits)*	torque measured & processed motor speed (16 bits)		accelerator position (8 bits)	*Brake Switch*
12V Power Ack Inverter	clutch pressure control & torque command (16 bits)	*Shutdown*	transaction clutch line pressure (8 bits)	*Shift Lever (PRNDL)*	cylinder and line break pressure (16 bits)
high and low contactor control (16 bits)	*Key Switch*	torque measured & processed motor speed (16 bits)		accelerator position (8 bits)	
12V Power Ack I/M Contr.	clutch pressure control & torque command (16 bits)	*Inverter Temperature Status*	transaction clutch line pressure (8 bits)	*Speed Control*	cylinder and line break pressure (16 bits)
traction battery average and max. temperature (16 bits)	*FWD/REV*	torque measured & processed motor speed (16 bits)		accelerator position (8 bits)	*Brake Switch*
Interlock	clutch pressure control & torque command (16 bits)		transaction clutch line pressure (8 bits)	*Brake Mode (Parallel/Split)*	cylinder and line break pressure (16 bits)
high and low contactor control (16 bits)	*Idle*	torque measured & processed motor speed (16 bits)		accelerator position (8 bits)	
DC/DC Converter Current Control (8 bits)	clutch pressure control & torque command (16 bits)		transaction clutch line pressure (8 bits)	*SOC Reset*	cylinder and line break pressure (16 bits)
12V Power Relay	*Shift in Progress*	torque measured & processed motor speed (16 bits)		accelerator position (8 bits)	*Brake Switch*
	clutch pressure control & torque command (16 bits)		transaction clutch line pressure (8 bits)		cylinder and line break pressure (16 bits)
high and low contactor control (16 bits)	*Main Contactor Acknowledge*	torque measured & processed motor speed (16 bits)		accelerator position (8 bits)	
	clutch pressure control & torque command (16 bits)		transaction clutch line pressure (8 bits)		cylinder and line break pressure (16 bits)
		torque measured & processed motor speed (16 bits)		accelerator position (8 bits)	*Brake Switch*

Figura 5.10: Ciclo de sistema para mapeamento do benchmark SAE.

Como estudo de caso da plataforma CASCA foi escolhido o mapeamento de um benchmark da Sociedade dos Engenheiros Automotivos (SAE) apresentado em (TINDELL; BURNS, 1995). Trata-se de um sistema distribuído com 8 nodos: painel de instrumentação (instrumentation pannel), bateria (battery), controle do veículo (vehicle controller), inversor (inverter), controle de transmissão (transmission control), entradas

do condutor (driver inputs) e freios (brakes). O painel de instrumentação é o único que não transmite mensagens logo não existe a necessidade de alocar um slot para ele.

No total são 53 requisitos de comunicação periódicos e aperiódicos os quais foram alocados em um ciclo de sistema da plataforma CASCA ilustrado na Figura 5.10. A seqüencia de 7 slots em 1 ciclo TDMA está representada na horizontal enquanto que o ciclo de sistema avança na vertical, de cima para baixo, de forma que o nodo da bateria é o primeiro a transmitir e o nodo de freios o último.

O P_{min} do benchmark é igual a 5ms contudo o tamanho de 1 ciclo TMDA foi ajustado para 2.5ms devido ao fato de um mesmo nodo possuir 2 ou mais requisitos do tipo <P_{min}, L_i>. O nodo de freios, por exemplo, deve transmitir as mensagens 9, 8 e 18 com requisitos <5ms,8>, <5ms,8> e <20ms,1> respectivamente.

Os ciclos TDMA duram 2.5ms enquanto que o ciclo de sistema completo, composto por 20 ciclos TDMA tem uma duração total de 20$\times$2.5ms=50ms. As mensagens são todas de tamanho igual ou menor que 8 bits logo algumas foram configuradas para transmissão em um mesmo quadro CAN dentro de um único slot como é caso da pressão de cilindro de da pressão de linha do freio (**cylinder and line break pressure**). Em caractere itálico são indicadas as mensagens aperiódicas que recebem um slot dedicado para serem transmitidas na medida em que são produzidos. Alguns requisitos periódicos como por exemplo status da bateria (**traction battery voltage & current**) de 100ms foram diminuídos para 50ms para não extrapolar o número de ciclos TDMA a serem configurados. Como consequência, a cada ciclo de sistema uma mensagem com período de 50ms é transmitida somente uma vez.

Com base nestas decisões de projeto tem-se que:

- `cNumerOfNodes`=7

  Considerando somente os nodos que transmitem mensagens.

- $\tau_{slot} \simeq \frac{2.5}{7}$ms

  O tamanho de 1 slot de ser aproximadamente igual ao tamanho de 1 ciclo TDMA dividido pelo número de nodos do sistema.

- f_{ck}^{-1}=20ns

  Tamanho de 1 microtick considerando que o oscilador da plataforma é de 50Mhz.

- φ=2

  Existem no máximo 2 slots vazios consecutivos o que ocorre no último ciclo TDMA.

Com estas informações é possível determinar os demais parâmetros de configuração da plataforma. O intervalo de sincronização (pior caso) é dado por:

$$R_{int}=(\varphi+1)\tau_{slot}=(2+1)\frac{2.5}{7}10^{-3}=1.07\text{ms}$$

A partir dos resultados práticos obtidos de protótipos verificou-se que para este valor de R_{int} a CASCA garante sincronização interna com precisão de 1μs, logo:

$$\texttt{cMicrotickMax} \geq 1\mu\text{s}.f_{ck}$$

$$\texttt{cMicrotickMax} \geq 50$$

Resta agora determinar o tamanho de 1 ciclo TDMA como um número inteiro de macroticks múltiplo de 7 – número de nodos ativos do sistema. Tem-se em princípio que:

$$\texttt{cMacrotickMax}=2.5\text{ms}(\texttt{cMicrotickMax})^{-1}fck$$

Fazendo `cMicrotickMax`=50 a equação acima resulta em:

$$\texttt{cMacrotickMax}=2500$$

Contudo, $\frac{2500}{7} = 357.14$ logo 2500 não é múltiplo inteiro de 7. Neste acaso adota-se o múltiplo mais próximo, ou seja:

$$\texttt{cMacrotickMax}=357\times 7=2499$$

O pior caso para o tempo de resposta de uma transmissão assíncrona, ou seja, o máximo de tempo que uma mensagem com esta característica espera para ser transmitida, acontece quando a solicitação é feita após o início do slot 4 do último ciclo TDMA. A transmissão é adiada até o próximo buraco que acontece após transcorridos 41 slots de tempo. Em relação ao tráfego periódico, as latências são praticamente fixas pois sua variação está limitada à precisão da base de tempo global, que neste estudo de caso é menor que 1μs.

6 VALIDAÇÃO E RESULTADOS

6.1 Simulação

A simulação do bloco lógico foi realizada utilzando a ferramenta **ModelSim** da **Mentor Graphics** com o objetivo de verificar o correto funcionamento da plataforma proposta. Através da simulação foi possível extrair os seguintes parâmetros essenciais para a síntese:

1. Tempo de processamento total do núcleo dentro de 1 slot – necessário para calcular o tamanho mínimo do slot de tempo possível de ser sintetizado.

2. Latências de entrada e saída – utilizado na definição do atraso de propagação a ser compensado no momento da leitura indireta.

Diversas baterias de simulação foram feitas no intuito de testar blocos individuais. Para cada entidade VHDL foi criado um testbench específico para validar seu funcionamento antes de integrá-lo junto com os demais. Contudo, neste capítulo serão apresentados somente os resultados mais significativos relativos à simulação global que fornecem os parâmetros essenciais para síntese.

Na simulação global foi criado um testbench que instancia dois controladores CASCA e o ciclo de sistema foi configurado com os seguintes parâmetros:

- `cNumberOfNodes` = 3

  O número de nodos **configurados** é maior que o número de nodos **instanciados** no testbench logo, existe sempre 1 slot vazio a cada ciclo TDMA.

- `cMicrotickMax` = 100

  Tamanho de 1 macrotick configurado como 100 ciclos de máquina.

- `cMacrotickMax` = 600

  Tamanho de 1 ciclo TDMA. Neste caso, o tamanho de 1 slot, em unidades de macrotick, é igual à:

  `cMacrotickMax`/`cNumberOfNodes`=200

O tempo de processamento do núcleo consiste no número total de ciclos de máquina gastos na execução das rotinas de tratamento de interrupções da plataforma durante as

fases de pré-transmissão (PSP) e pós-recepção (PRP) dentro de cada slot de tempo. O conhecimento destes parâmetros é fundamental para que se possa determinar o tamanho mínimo de slot que por sua vez está diretamente relacionado ao limite de performance da plataforma. A Figura 6.1 mostra uma janela de simulação onde foram incluídos os sinais `TXD` de cada nodo (`txd_i` e `txd_i1`), 2 registradores internos do bloco TDMA Clock responsáveis pela contagem de macroticks e slots ao longo de um ciclo TDMA, além do sinal auxiliar `mcu_busy` utilizado para monitorar o contador de programa do núcleo. Quando o núcleo está ocioso seu código fica em laço infinito, aguardando nova interrupção, percorrendo endereços de memória bem conhecidos. O sinal `mcu_busy` foi então criado somente para fins de simulação no intuito facilitar a visualização da parcela de tempo que o núcleo está ou em estado de espera (‘1’) ou ocupado no atendimento de uma interrupção (‘0’). A lógica de atribuição deste sinal é função unicamente do valor do `PC`.

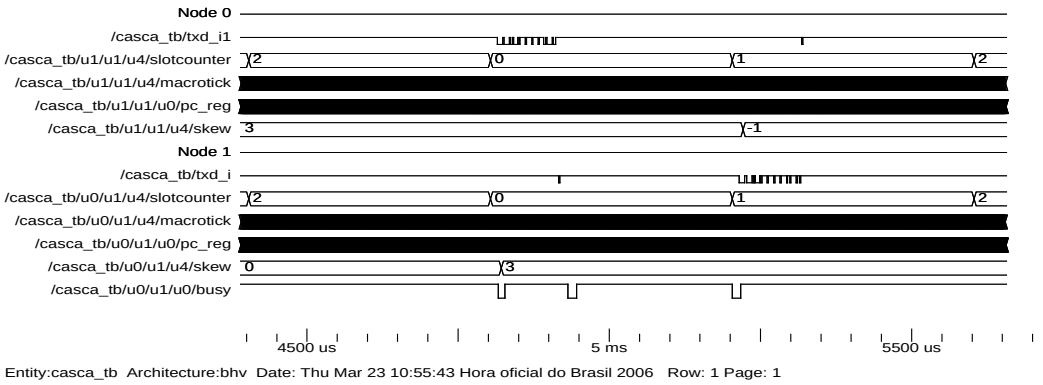

Figura 6.1: Simulação do controle de acesso TDMA.

Os resultados extraídos da simulação são mostrados na tabela 6.1. O tempo de processamento total do núcleo é igual à soma das fases de pré-transmissão (PSP) e pós-recepção (PRP). As latências de saída (Δ_{TX}) e de entrada (Δ_{RX}) foram contabilizadas juntas, medindo a distância em ciclos de máquina entre o início do slot de transmissão no lado transmissor e o momento em que a leitura indireta é feita no lado receptor. Como conseqüência da execução de algumas rotinas em software não foi possível reduzir a variação das latências do protocolo à zero, isto é:

$$\Delta_{TX.max} \neq \Delta_{TX.min}$$
$$\Delta_{RX.max} \neq \Delta_{RX.min}$$

Em razão disto o valor somado de Δ_{TX}+Δ_{RX} apresentado na tabela é o valor médio. Esta variabilidade tem impacto direto na precisão do algoritmo de sincronização pois uma parcela de erro adicional teve de ser considerada na estimativa de δ_{max} conforme já destacado no capítulo 5.

parâmetro	# ciclos
Δ_{TX}+Δ_{RX}	900
PRP	728
PSP	936

Tabela 6.1: Resultados da simulação.

O barramento CAN foi simulado como uma porta AND com atraso elétrico zero. Esta AND recebe como entrada os pinos `TXD` de ambos os nodos e conecta sua saída nos pinos `RXD` dos mesmos. Assim, o atraso de propagação observado é resultado somente das latências de atendimento de interrupções da plataforma. A parcela que resta a ser considerada, o atraso do sinal no meio físico Δ_{bus}, foi extraído diretamente do protótipo com auxílio de um osciloscópio.

As Figuras 6.2 e 6.3 mostram os dois principais momentos da execução do algoritmo de sincronização Daisy-Chain. No nodo 0 (lado transmissor) uma interrupção é disparada no início do seu respectivo slot para que o núcleo inicie a transmissão do quadro CAN. Quando o início do quadro é detectado no nodo 1 (lado receptor) uma segunda interrupção é disparada fazendo com que o núcleo envie um comando `GET_SKEW` para o bloco TDMA Clock que por sua vez infere o desvio entre ambas as bases de tempo a partir da diferença entre o instante esperado e o valor atual da base local. O resultado desta diferença é colocado no registrador `skew` o qual é carregado com o valor 6. Uma vez que na simulação efeito de ρ não foi considerado, este desvio é resultante de um defasamento inicial e do erro associado ao atraso de propagação configurado inicialmente.

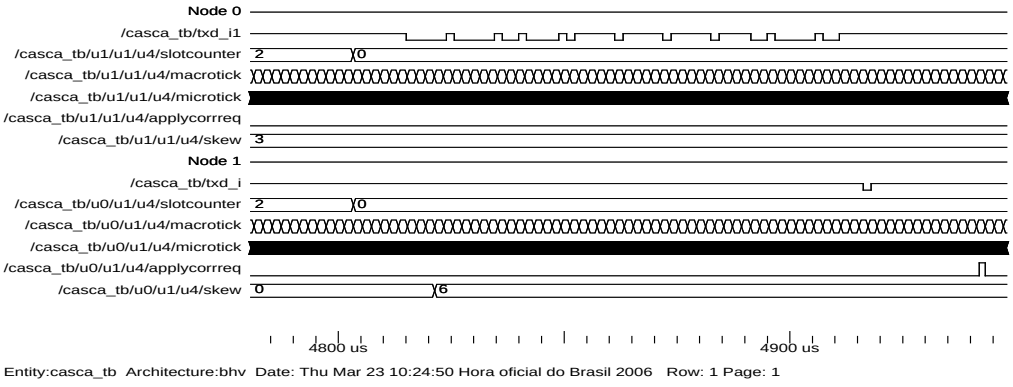

Figura 6.2: Sincronização de relógios: instante de leitura.

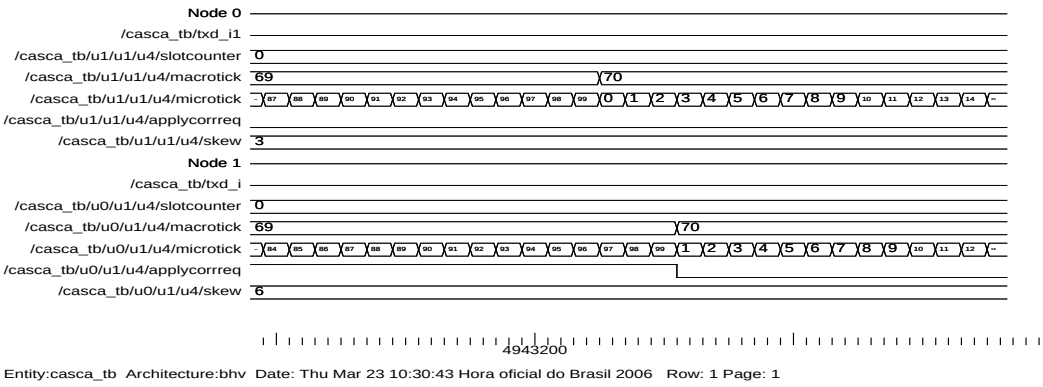

Figura 6.3: Sincronização de relógios: instante de correção.

Após o correto recebimento da mensagem o núcleo do nodo receptor envia o comando `APPLY_CORR_TERM` para que o TDMA Clock aplique o fator de correção na base de tempo local. Observa-se na Figura 6.3 que o fator de correção representa somente uma parcela do desvio inferido. O contador de microtick é adiantado somente por 1 unidade de tempo ao invés de 6.

6.2 Prototipação

O objetivo da construção dos protótipos foi basicamente o de verificar na prática o correto funcionamento das seguintes funcionalidades da plataforma:

1. Mecanismo de acesso não-destrutivo do protocolo CAN original.
2. Algoritmo de sincronização Daisy-Chain.
3. Esquema de acesso TDMA.
4. Mecanismo de inicialização da base de tempo.

Foram utilizadas 3 placas de desenvolvimento, cada uma provida de um dispositivo FPGA Xilinx Spartan2E e interface JTAG através da qual os controladores de rede puderam ser inteiramente sintetizados em lógica programável. Para que as placas pudessem comunicar através de um mesmo barramento serial foram construídas também 3 placas de interface analógica utilizando dispositivos transdutores CAN 82C250 (Figura 6.4). Os pinos TXD e RXD são conectados aos pinos de I/O do FPGA por intermédio de divisores de tensão resistivos e resistores de pull-up para garantir níveis corretos de saturação e aberto dos transistores de entrada, tanto do lado do FPGA que utiliza drivers CMOS quanto do lado do transdutor que utiliza lógica TTL. No caso dos pinos CAN_H e CAN_L estes são diretamente conectados ao meio físico para transmissão na forma analógica diferencial.

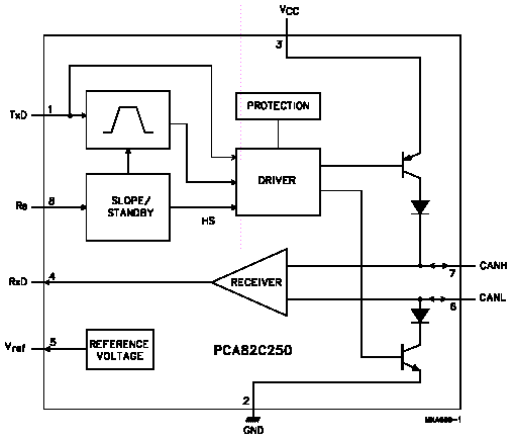

Figura 6.4: Organização interna e pinagem do transdutor 82C250.

O primeiro protótipo é formado por apenas dois FGPAs nos quais foram sintetizados somente os blocos BSP e BTL que implementam as camadas 1 e 2 do protocolo CAN nativo. No intuito de forçar colisões o bloco BSP foi sintetizado com sua flag de transmissão interna presa em ‘1’. Deste modo, a máquina de estados do protocolo entende que existe sempre uma mensagem a ser enviada no buffer de saída fazendo com que a cada tentativa de acesso ocorra uma colisão entre os dois nodos. O resultado esperado foi corretamente verificado na prática conforme apresentado na Figura 6.5 onde uma imagem capturada de osciloscópio mostra o comportamento dos sinais de transmissão serial de cada nodo. Na imagem da esquerda é mostrado a transmissão de um quadro onde a fase de arbitração é claramente observável ao passo que à direita é mostrado uma imagem com janela de tempo maior na qual é possível visualizar uma seqüencia de 4 transmissões sucessivas com conflito de acesso. O sinal de baixo provém do pino TXD do nodo com maior prioridade o qual sempre ganha o acesso sem mesmo perceber a colisão enquanto que acima é

o sinal TXD do nodo memos prioritário que perde o barramento sempre no momento em que envia um bit dominante e lê um recessivo no meio, tornando-se receptor do quadro. Ao final da transmissão o receptor transmite um bit dominante de confirmação conforme previsto na norma do protocolo.

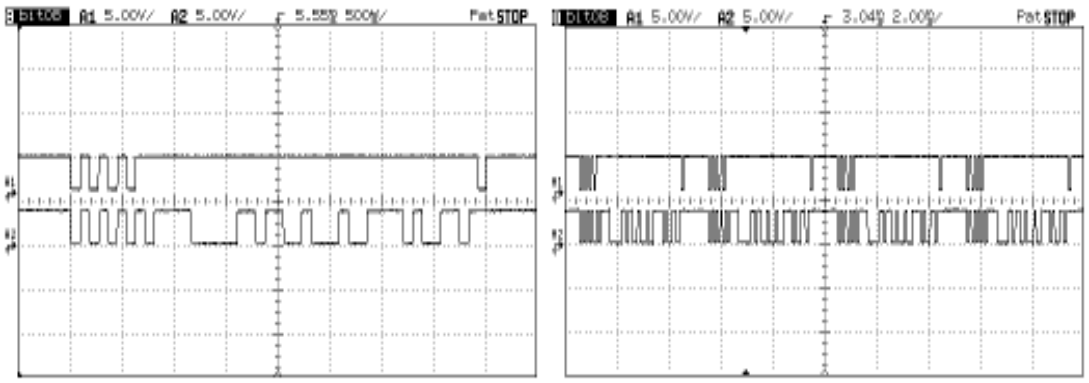

Figura 6.5: Mecanismo de resolução de colisões não-destrutivo.

A partir deste primeiro protótipo também foi extraído a componente Δ_{bus} do atraso de propagação a ser compensado no momento de cada leitura indireta. A Figura 6.6 mostra uma imagem com o comportamento dos pinos TXD e RXD dos nodos com maior distância relativa entre um e outro. As duas placas estão separadas por aproximadamente 8 metros de cabo resultando em um atraso elétrico de 164ns. Para a menor distância relativa entre dois nodos o atraso elétrico observado foi de 64ns logo:

$$\Delta_{bus.max} = 164\text{ns}$$
$$\Delta_{bus.min} = 64\text{ns}$$

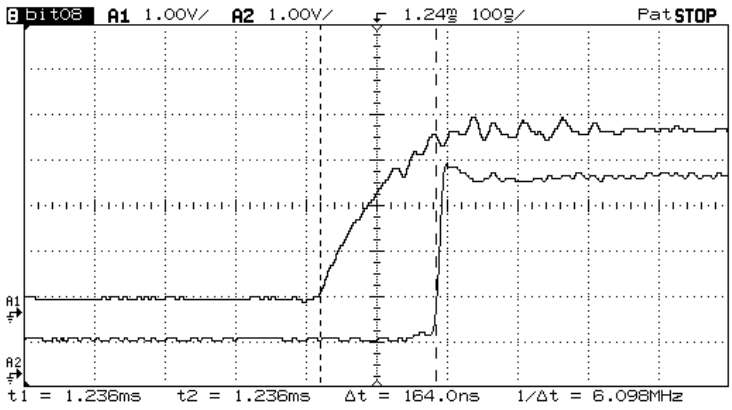

Figura 6.6: Atraso de propagação de sinal no barramento.

Assume-se então **para este meio físico** um atraso elétrico igual a:

$$\Delta_{bus} = \frac{164+64}{2}\text{ns} = 114\text{ns}$$

e um erro associado de:

$$\varepsilon = \frac{164-64}{2}\text{ns} = 50\text{ns}$$

Tem-se então que o atraso de propagação total a ser compensado é:

$$\Delta_{TX}+\Delta_{RX}+\Delta_{bus}$$

onde Δ_{TX} e Δ_{RX} foram extraídos da simulação.

O segundo protótipo consiste em um sistema distribuído elementar utilizando os 3 nodos disponíveis no laboratório, onde cada um foi configurado com um ciclo de sistema formado por apenas um ciclo TDMA com 5 slots de tempo cada. O objetivo principal sendo verificar o correto funcionamento do mecanismo TDMA e da inicialização da base de tempo a precisão foi extrapolada para 2μs. Sabendo que o oscilador da plataforma é de 50Mhz o tamanho de 1 macrotick foi configurado como sendo 100 pulsos de máquina, ou seja, `cMicrotickMax`=100.

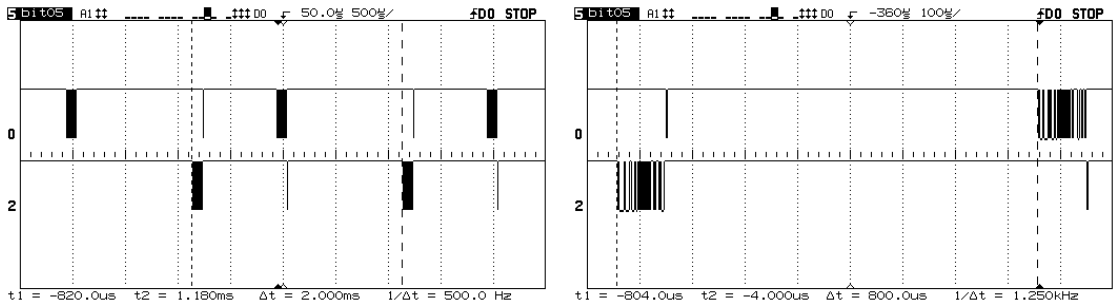

Figura 6.7: Esquema TDMA com 2 nodos ativos.

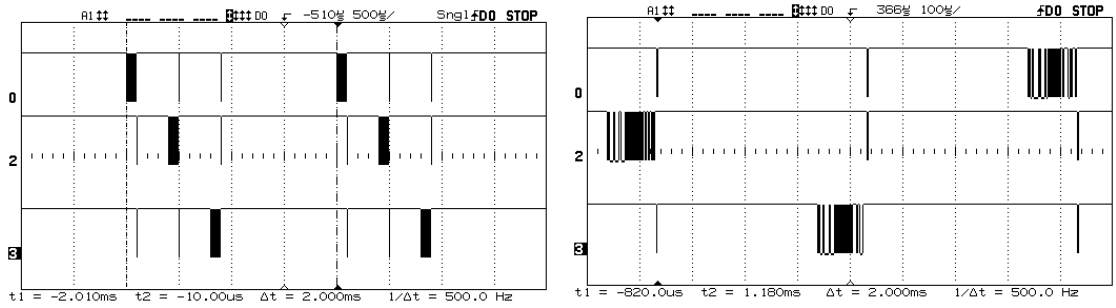

Figura 6.8: Esquema TDMA com 3 nodos ativos.

Nas imagens de osciloscópio da Figura 6.7 tem-se o sistema operando com 2 nodos transmitindo quadros com mensagens sem conteúdo prático somente no intuito de mostrar que o mecanismo TDMA funciona corretamente. Ambos os nodos conseguem corrigir suas bases de tempo e se manter sincronizados de forma a garantir a multiplexagem no tempo de acesso ao meio. A seguir o terceiro nodo é energizado e integra-se à base de tempo já existente, transmitindo em seu slot de tempo exclusivo tal como mostrado nas imagens da Figura 6.8. A partir de então qualquer um dos 3 nodos podem sair e entrar novamente na comunicação sem interromper a comunicação entre os demais, o que caracteriza de fato uma base de tempo essencialmente distribuída.

Os resultados da síntese do bloco lógico extraído da ferramenta **Project Navigator** são mostrados no gráfico da Figura 6.9 em termos de utilização de recursos de lógica programável do FPGA Xilinx Spartan2E. Cabe destacar que a implementação do protocolo CAN representa aproximadamente 30% do tamanho do circuito e o restante sendo

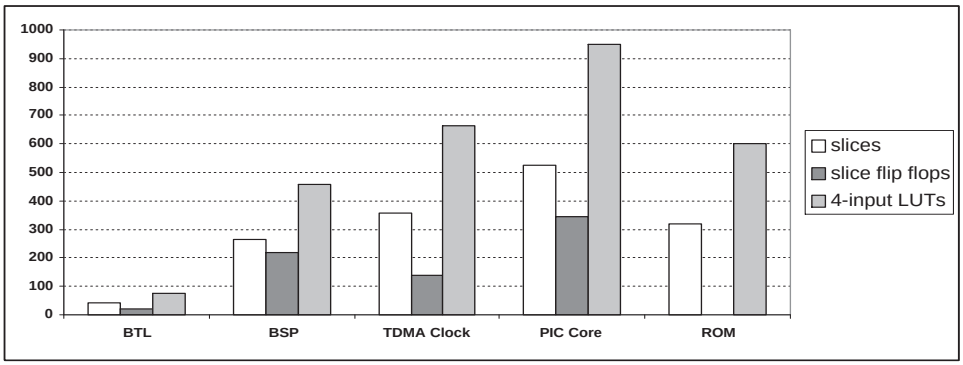

Figura 6.9: Resultados da síntese em FPGA.

utilizado na síntese das funções primitivas de sincronização, criação da base de tempo e controle de acesso TDMA.

6.3 Avaliação da Estabilidade do Algoritmo de Sincronização

Esta etapa teve como objetivo avaliar a estabilidade do algoritmo Daisy-Chain no que se refere à precisão da base de tempo em função de algumas características que variam conforme a definição do ciclo de sistema. Para isto uma UART unidirecional foi sintetizada dentro do bloco TDMA Clock com a função de exportar o conteúdo do registrador `SKEW` de um dos nodos toda vez que uma nova leitura é feita. Desta forma foi possível capturar o comportamento da algoritmo DCA em tempo de operação pela interface RS232 de uma máquina desktop.

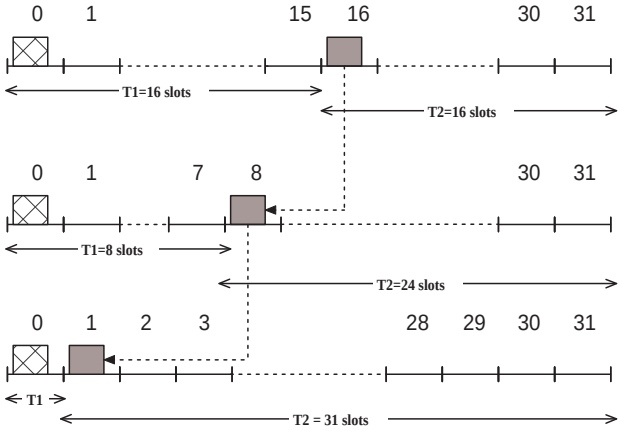

Figura 6.10: Modificação do espaço entre duas sincronizações.

Na Figura 6.11 são mostrados dois gráficos com constelações de pontos que representam os desvios inferidos ao longo do tempo em um protótipo configurado primeiramente com 1 ciclo TDMA de 32 slots. O eixo vertical representa o desvio inferido no momento da leitura em unidades de microtick enquanto que o eixo horizontal representa a evolução do sistema na linha do tempo físico. Sabendo que a fonte primária da plataforma tem freqüência de 50Mhz então cada unidade de microtick tem o tamanho de 20ns. Os valores negativos indicam que o nodo onde a UART foi sintetizada, chamado **nodo fonte**, está adiantado em relação aos nodos flutuantes, ou seja, nodos que “saltam de um slot para

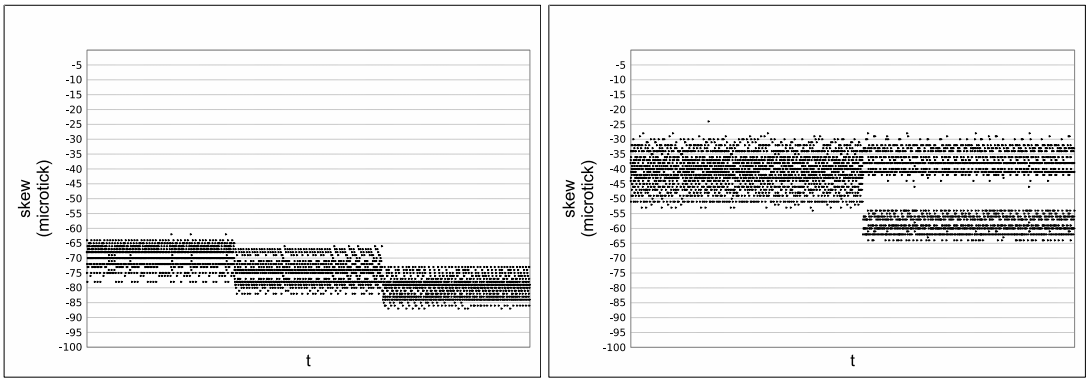

Figura 6.11: Desvios obtidos pela interface RS232: 32 slots.

outro.

Ainda na Figura 6.11 o gráfico da esquerda mostra os desvios capturados em um sistema onde apenas 2 nodos comunicam-se. O nodo fonte permanece fixo no slot 0 enquanto que o nodo flutuante entra transmitindo no slot 16, depois salta para o 8 e finalmente para o 1, fazendo variar o tamanho máximo de R_{int} de 16 para 24, depois para 31 slots, conforme ilustrado no esquema da Figura 6.10. À direita o gráfico mostra os resultados após o terceiro nodo disponível ter sido anexado ao sistema integrando-se à base de tempo dos demais. Neste caso o nodo fonte continua no slot 0 e os outros começam transmitindo nos slots 10 e 22 mas depois saltam para os slots 1 e 2, fazendo variar novamente o tamanho máximo do intervalo de sincronização durante o qual nenhuma correção é feita.

Em ambos os casos descritos acima acontece um aumento do desvio máximo, ou seja, a precisão da base de tempo se deteriora na medida que R_{int} aumenta. Mais importante ainda, os resultados práticos mostram que para um mesmo número de nodos a relação entre δ_{max} e R_{int} é praticamente **linear**, conforme previsto na formalização do algoritmo Daisy-Chain. Outra característica observada é que a precisão também é sensível ao número de transmissões por ciclo TDMA. Com 2 transmissões por ciclo o pior caso para δ_{max} chegou a 87 microticks (1.74μs) ao passo que com 3 nodos o desvio máximo diminuiu para 65 microticks (1.3μs).

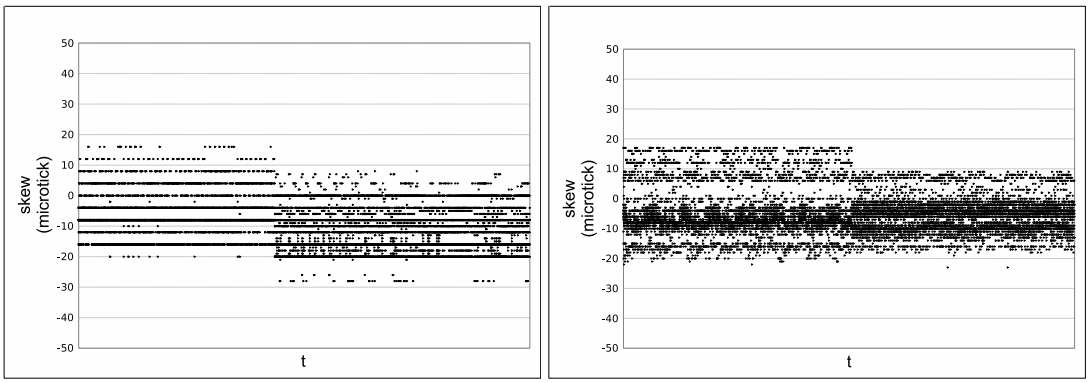

Figura 6.12: Desvios obtidos pela interface RS232: 12 e 6 slots.

É importante salientar que os desvios capturados desta primeira configuração com 32 slots e apenas dois nodos transmitindo nos slots 0 e 1 representam o pior caso de uso. Assim, pode-se afirmar que um sistema distribuído construído sobre a CASCA possui

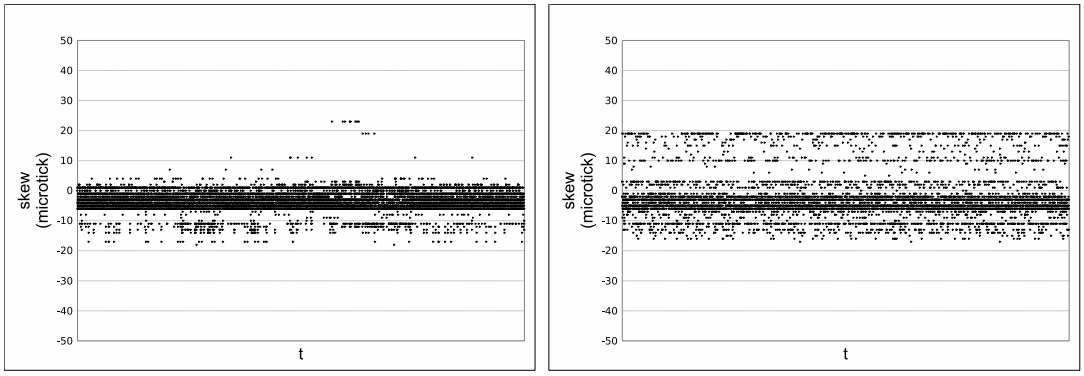

Figura 6.13: Desvios obtidos pela interface RS232: 3 e 2 slots.

uma base de tempo global com granularidade máxima de 1.74μs. Resta portanto saber a precisão máxima atingível. Nos ensaios subseqüentes o protótipo foi configurado com um ciclo TDMA menor visando diminuir ainda mais o valor de R_{int} e aumentar o número de sincronizações por ciclo.

Na Figura 6.12 o gráfico da esquerda apresenta resultados obtidos de um ciclo TDMA com 12 slots no qual o nodo fonte está fixo no slot 0 e os flutuantes saltam dos slots 4 e 8 para 1 e 2, enquanto que no gráfico da direita existem 6 slots por ciclo TDMA e os nodos flutuantes saltam de 2 e 4 para 1 e 2.

Os resultados finais são mostrados na Figura 6.13. À esquerda resultados de um ciclo TDMA com 3 slots com todos os 3 nodos transmitindo nos slots 1, 2 e 3 quanto que à esquerda o ciclo TDMA possui somente 2 slots. Neste caso um nodo foi retirado e os dois nodos restantes transmitem nos slots 0 e 1 de maneira alternada. Nos dois casos a precisão verificada na prática foi de 20 microticks, isto é, 0.4μs.

7 CONCLUSOES & CONSIDERAÇÕES FINAIS

O presente texto apresentou a proposta e implementação de uma plataforma de comunicação para aplicações críticas em sistemas distribuídos a qual foi designado o nome de CASCA “Communication Architecture for Safety-Critical Applications”. Foram abordados na teoria e na prática os principais desafios relacionados ao projeto de um esquema de comunicação TDMA, utilizando para isto um meio físico de difusão serial compartilhado. Foi mostrado através de resultados em protótipos que é possível manter uma base de tempo essencialmente distribuída com precisão menor do que 1μs mesmo usando dispositivos osciladores de baixo custo. No que se refere à criação da base de tempo, CASCA foi dotada de um procedimento extremamente simples e eficiente pois se beneficia do mecanismo anti-colisão não-destrutivo do protocolo CAN original para lidar com eventuais conflitos de acesso durante a fase inicial.

A sincronização de relógios na plataforma proposta é garantida pela execução do algoritmo Daisy-Chain o qual dispensa o armazenamento temporário de leituras tornando mais simples e mais robusta a sua implementação em comparação à outros métodos que são atualmente aplicados em protocolos comerciais. Tirando proveito do mecanismo TDMA, no qual todas as transmissões são previamente agendadas, as leituras de relógio são feitas de maneira indireta logo não existe de fato a necessidade de incluir valores de relógio no corpo das mensagens. O ajuste do relógio local é feito somente por correção de fase sempre quando um quadro é transmitido no barramento. Ajustes acontecem com alta freqüência de forma que a ocorrência de falhas esporádicas no procedimento de leitura, cujo efeito é amenizado pela função de convergência elementar utilizada, não comprometem a condição de sincronização.

Sendo CASCA construída sobre o protocolo CAN original não existe de fato a necessidade de projetar as camadas física e de enlace a partir do zero. Controladores de rede podem ser implementados com o reuso de módulos de hardware já existentes como é o caso do TTCAN, também um extensão time-triggered de CAN. Ao contrário do TTCAN, em CASCA não existe a diferenciação entre mestres e escravos. A base de tempo possui uma característica essencialmente distribuída o que torna um sistema distribuído mais resiliente pelo fato de não existir um ponto crítico de falhas que representa o mestre TTCAN.

A validação de CASCA consistiu na prototipação de um sistema distribuído elementar utilizando placas de desenvolvimento em dispositivos FPGA de baixo custo, ou seja, com baixa disponibilidade de recursos de lógica programável. A utilização de CASCA está voltada ao projeto de sistemas distribuídos seguros, embarcados ou industriais, destinados à aplicações críticas as quais requerem altos níveis de confiabilidade. O modelo de comunicação time-triggered, a base de tempo distribuída e a possibilidade de replicação do meio físico oferecem suporte ao uso de técnicas de tolerância à falhas tais como o

mapeamento de unidades redundantes. A alta precisão da base de tempo, a fácil e rápida detecção de falhas e a autonomia na comunicação oferecida pelo modelo time-triggered torna possível o gerenciamento de unidades redundantes na plataforma, fazendo com que este aspecto seja transparente para a camada da aplicação.

No decorrer do texto mostrou-se também como configurar um sistema baseado em TDMA a partir de um conjunto de requisitos de transmissão periódicos. Com base na metodologia proposta foi possível mapear todos os requisitos do benchmark da SAE (Sociedade dos Engenheiros Automotivos) provando que CASCA oferece performance, e.g. vazão efetiva suficiente para aplicações no domínio embarcado.

Como trabalhos futuros, sugere-se os a prototipação de subsistemas redundantes construídos sobre canais de comunicação replicados, o projeto de dispositivos de guarda para proteger o(s) barramento(s) contra falhas no núcleo, além do desenvolvimento de ferramentas de geração automática de definição do ciclo de sistema partindo de um conjunto de requisitos de comunicação.

REFERÊNCIAS

ALBERT, A. Comparison of Event-Triggered and Time-Triggered Concepts with Regard to Distributed Control Systems. **Embedded World**, [S.l.], p.235–252, 2004.

AVIZIENIS, A. Design of Fault-Tolerant Computers. In: AFIPS CONFERENCE, 1967. **Proceedings...** [S.l.: s.n.], 1967. p.733–743.

AVIZIENIS, A. Dependable Computing Depends On Structured Fault Tolerance. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE RELIABILITY ENGINEERING, 6., 1995. **Proceedings...** [S.l.: s.n.], 1995. p.158–168.

AVIZIENIS, A. **Infrastructure-Based Design of Fault-Tolerant Systems – How to Get High-Confidence Computing for All**. [S.l.]: UCLA Computer Science Department, University of California, 1997. Research Report.

AVIZIENIS, A. Basic Concepts And Taxonomy Of Dependable And Secure Computing. **IEEE Transactions on Dependable and Secure Computing**, [S.l.], v.1, p.11–33, 2004.

BERGERON, J. **Writing Testbenches Functional Verification of HDL Models**. [S.l.]: Kluwer Academic Publishers, 2001.

BERWANGER, J.; PELLER, M.; GRIESSBACH, R. Byteflight – A New High-Performance Data Bus System for Safety-Related Applications. In: BMW AG EE-211 DEVELOPMENT SAFETY SYSTEM ELECTRONICS, 2000. **Proceedings...** [S.l.: s.n.], 2000.

BOWLES, J. B. A Survey of Reliability - Prediction Procedures For Microelectronic Devices. **IEEE Transactions on Reliability**, [S.l.], v.41, p.651–663, 1992.

BRIDAL, O.; SNEDSBØL, R.; JOHANSSON, L.-A. On the Design of Communication Protocols for Safety-Critical Automotive Applications. In: IEEE VEHICULAR TECHNOLOGY CONFERENCE, 44., 1994. **Proceedings...** [S.l.]: IEEE, 1994. v.2, p.1098–1102.

BROSTER, I.; BURNS, A. Timely use of the CAN Protocol in Critical Hard Real-time Systems with Faults. In: EUROMICRO CONFERENCE ON REAL-TIME SYSTEMS, ECRTS, 13., 2001. **Proceedings...** [S.l.: s.n.], 2001.

CARVALHO, F. C.; PEREIRA, C. E.; FREITAS, E. P.; Silva Jr., E. T. A Practical Implementation of the Fault-Tolerant Daisy-Chain Clock Synchronization Algorithm on CAN. In: DESIGN, AUTONATION AND TEST CONFERENCE, DATE, 9., 2006. **Proceedings...** [S.l.: s.n.], 2006.

CARVALHO, F. C.; PEREIRA, C. E.; FREITAS, E. P.; Silva Jr., E. T. A Time-Triggered Controller Area Network Platform with Essentially Distributed Clock Synchronization. In: IFAC SYMPOSIUM ON INFORMATION CONTROL PROBLEMS IN MANUFACTURING, 12., 2006. **Proceedings...** [S.l.: s.n.], 2006.

CIA. **CAN Specification 2.0**. [S.l.]: CAN in Automation, 1999.

DEFAGO, X.; SCHIPER, A.; URBAN, P. Total Order Broadcast and Multicast Algorithms – Taxonomy and Survey. **ACM Computing Surveys**, [S.l.], v.36, n.4, p.372–421, 2004.

DILGER, E.; FüHRER, T.; MüLLER, B.; POLEDNA, S. **The X-By-Wire Concept – Time Triggered Information Exchange and Fail Silence Support by new System Services**. Disponível em: <http://www.vmars.tuwien.ac.at/projects/xbywire/projects/new-bosch.htm> Acesso em: set. 2005.

FELSER, M. Ethernet TCP/IP in Automation – A Short Introduction to Real-Time Requirements. In: IEEE INTERNATIONAL CONFERENCE ON EMERGING TECHNOLOGIES AND FACTORY AUTOMATION, 8., 2001. **Proceedings...** [S.l.: s.n.], 2001. v.2, p.501–504.

FLEXRAY. **FlexRay Communications System Protocol Specification Version 2.0**. [S.l.]: FlexRay Consortium, 2004.

FUHRER, T.; MüLLER, B.; DIETERLE, W.; HARTWICH, F.; HUGEL, R.; WALTHER, M.; GMBH, R. B. Time–Triggered Communication on CAN. In: INTERNATIONAL CAN CONFERENCE, 7., 2000. **Proceedings...** [S.l.: s.n.], 2000.

HOYME, K.; DRISCOLL, K. SAFEbus. **IEEE Aerospace and Electronic Systems Magazine**, [S.l.], v.8, n.3, p.52–78, March 1993.

JANG, S.-H.; PARK, T.-J.; HAN, C.-S. A Control of Vehicle Using Steer-By-Wire System with Hardware-in-the-Loop-Simulation System. In: IEEE/ASME INTERNATIONAL CONFERENCE ON ADVANCED INTELLIGENT MECHATRONICS, 2003. **Proceedings...** [S.l.: s.n.], 2003. p.389–394.

KOPETZ, H.; ADEMAJ, A.; HANZLIK, A. Integration of Internal and External Clock Synchronization by the Combination of Clock-State and Clock-Rate Correction in Fault-Tolerant Distributed Systems. In: IEEE INTERNATIONAL REAL-TIME SYSTEMS SYMPOSIUM, 25., 2004. **Proceedings...** [S.l.: s.n.], 2004. p.415–425.

KOPETZ, H.; BAUER, G. The Time-Triggered Architecture. **Proceedings of the IEEE**, Piscataway, v.91, n.1, p.112–126, 2003.

KOPETZ, H.; OCHSENREITER, W. Clock Synchronization In Distributed Real-Time Systems. **IEEE Trans. Comput.**, Washington, DC, USA, v.36, n.8, p.933–940, 1987.

KWAK, B.-J.; SONG, N.-O.; MILLER, L. E. Analysis of the Stability and Performance of Exponential Backoff. In: IEEE WIRELESS COMMUNICATIONS AND NETWORKING CONFERENCE, 2003. **Proceedings...** [S.l.: s.n.], 2003.

LAMPORT, L. Using Time Instead of Timeout for Fault-Tolerant Distributed Systems. **ACM Trans. Program. Lang. Syst.**, New York, NY, USA, v.6, n.2, p.254–280, 1984.

LAMPORT, L.; MELLIAR-SMITH, P. M. Synchronizing Clocks in the Presence of Faults. **Journal of the ACM**, [S.l.], v.32, n.1, p.52–78, 1985.

LAMPORT, L.; SHOSTAK, R.; PEASE, M. The Byzantine Generals Problem. **ACM Transactions on Programming Languages and Systems**, [S.l.], p.382–401, July 1982.

LONN, H. Initial Synchronization of TDMA Communication in Distributed Real-Time Systems. In: IEEE INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS, 19., 1999. **Proceedings...** [S.l.: s.n.], 1999. p.370–379.

LONN, H. **The Fault Tolerant Daisy Chain Clock Synchronization Algorithm**. [S.l.]: Chalmers University of Technology, 1999. Research Report.

LONN, H.; SNEDSBØL, R. Synchronisation in Safety-Critical Distributed Control Systems. In: IEEE INTERNATIONAL CONFERENCE ON ALGORITHMS AND ARCHITECTURES FOR PARALLEL PROCESSING, 1995. **Proceedings...** [S.l.: s.n.], 1995.

MILLS, D. L. Internet Time Synchronization – The Network Time Protocol. **IEEE Transactions on Communications**, [S.l.], v.39, n.10, p.1482–1493, 1991.

MODBUS. **Application Protocol Specification V1.1a**. [S.l.: s.n.], 2004.

PALANQUE, P.; PATERNO, F.; WRIGHT, P. Designing User Interfaces for Safety-critical Systems. In: CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, 1998. **Proceedings...** [S.l.: s.n.], 1998.

PANKAJ, J. **Fault Tolerance In Distributed Systems**. New Jersey: Prentice Hall, 1994.

POLEDNA, S.; NIEDERSUSS, M.; KROISS, G. **TTPC - Fault-Tolerant Real-Time Communication with integrated High-Level Services**. [S.l.: s.n.], 2000.

PROFIBUS. **Profibus Standard DIN 19245 part I and II**. [S.l.: s.n.], 1992.

RAMANATHAN, P.; SHIN, K. G.; BUTLER, R. W. Fault-Tolerant Clock Synchronization in Distributed Systems. **Computer**, [S.l.], v.23, n.10, p.33–42, 1990.

ROSTAMZADEH, B.; LONN, H.; SNEDSBGL, R.; TORIN, J. DACAPO – A Distributed Computer Architecture for Safety-Critical Control Applications. In: INTELLIGENT VEHICLES SYMPOSIUM, 1995. **Proceedings...** [S.l.: s.n.], 1995. p.376–381.

RUSHBY, J. Bus Architectures For Safety-Critical Embedded Systems. In: EMSOFT, 2001. **Proceedings...** [S.l.: s.n.], 2001.

SCHMUCK, F.; CRISTIAN, F. Continuous Clock Amortization Need not Affect the Precision of a Clock Synchronization Algorithm. In: ACM SYMPOSIUM ON PRINCIPLES OF DISTRIBUTED COMPUTING, 9., 1990. **Proceedings...** [S.l.: s.n.], 1990. p.133–143.

SCHNEIDER, F. B. **A Paradigm for Reliable Clock Synchronization**. [S.l.]: Department of Computer Science, Cornell University, 1986. Research Report.

SONG, Y. Time Constrained Communication over Switched Ethernet. In: IFAC INTERNATIONAL CONFERENCE ON FIELDBUS SYSTEMS AND THEIR APPLICATIONS, 4., 2001. **Proceedings...** [S.l.: s.n.], 2001.

SOURCEBOOST IDE. Disponível em: <http://www.picant.com/c2c/c.html> Acesso em: ago. 2005.

SRIKANTH, T. K.; TOUEG, S. Optimal Clock Synchronization. **J. ACM**, New York, NY, USA, v.34, n.3, p.626–645, 1987.

SRINIVASAN, J.; LUNDQVIST, K. Real-Time Architecture Analysis – A COTS Perspective. In: DIGITAL AVIONICS SYSTEMS CONFERENCE, 21., 2002. **Proceedings...** [S.l.: s.n.], 2002. p.5D4–1–5D4–9.

SURI, N.; HUGUE, M. M.; WALTER, C. J. Synchronization Issues in Real-Time Systems. **Proceedings of the IEEE**, [S.l.], v.82, n.1, p.41–54, January 1994.

THAMBIDURAI, P.; PARK, Y.-K. Interactive Consistency With Multiple Failure Modes. In: SYMPOSIUM ON RELIABLE DISTRIBUTED SYSTEMS, 7., 1988. **Proceedings...** [S.l.: s.n.], 1988. p.93–100.

TINDELL, K.; BURNS, A. Guaranteeing Message Latencies on Control Area Network. **Control Engineering Practice**, [S.l.], v.3, n.8, p.1163–1169, 1995.

TOVAR, E.; VASQUES, F. Real-Time Fieldbus Communications Using Profibus Networks. **IEEE Transactions on Industrial Electronics**, [S.l.], v.46, n.6, p.1241–1251, 1999.

TTP/C. **Time-Triggered Protocol TTP/C High-Level Specification Document Protocol Version 1.1**. [S.l.]: TTA Group, 2003.

WELCH, J. L.; LYNCH, N. A New Fault-Tolerant Algorithm For Clock Synchronization. **Inf. Comput.**, Duluth, MN, USA, v.77, n.1, p.1–36, 1988.