Universidade Federal de Campina Grande
Centro de Ciências e Tecnologia
Departamento de Engenharia Elétrica

TRABALHO DE CONCLUSÃO DO CURSO

Aluno: Jadsonlee da Silva Sá
Matrícula: 29921229
Orientador: José Sérgio da Rocha Neto

Campina Grande
Fevereiro de 2005

Biblioteca Setorial do CDSA. Fevereiro de 2021.

Sumé - PB

Título:

Monitoração e Controle de um Sistema Distribuído Utilizando a Tecnologia CAN

Aluno:

Jadsonlee da Silva Sá

Orientador:

José Sérgio da Rocha Neto

Campina Grande

Fevereiro de 2005

Sumário

1 Introdução **1**

2 CAN **3**

2.1 Introdução 3
2.2 Aplicações 4
2.3 Características 5
2.4 Implementação de um Nó CAN 8
2.5 Método de Endereçamento 9
2.6 Processo de Transmissão e Recepção de Mensagens CAN 9
2.7 Processo de Arbitragem Não-Destrutivo 10
2.8 Formato das Mensagens CAN 12
2.9 Tipos de Mensagens 13
2.9.1 Pacote de Dados 13
2.9.2 Pacote de Requisição de Dados 15
2.9.3 Pacote de Erro 16
2.9.4 Pacote de Sobrecarga 18
2.10 Intervalo *Interframe* 19
2.11 Codificação 20
2.12 Métodos de Detecção e Sinalização de Erros 21
2.12.1 Erro de Inserção de *Bits* 21
2.12.2 Erro de *Bit* 22
2.12.3 Erro de CRC 23
2.12.4 Erro de Formato 24
2.12.5 Erro de Reconhecimento 24
2.13 Estados de Erro 24
2.14 *Bit Time* 25
2.15 Sincronismo 28
2.16 Meio de Transmissão 30
2.17 *Transceiver* 31
2.18 Tolerâncias de Falhas no Barramento 32

2.19 Arquitetura dos Controladores 32
2.19.1 Integração 32
2.19.2 Armazenamento de Mensagens 34
2.19.3 Filtro de Aceitação 35

3 Arquitetura e Descrição do Sistema Implementado **36**
3.1 Arquitetura do Sistema 36
3.2 Descrição do Funcionamento do Sistema 37
3.3 Programas dos Nós CAN 39
3.4 Arquitetura dos Nós CAN 42
3.5 Comunicação entre o Controlador MCP2510 e o ADuC812 43
3.5.1 Instrução de Leitura 44
3.5.2 Instrução de Escrita 45
3.5.3 Instrução de Leitura de *Status* 46
3.5.4 Instrução de Modificação de *Bit* 47
3.5.5 Instrução de *Reset* 48
3.5.6 Instrução de Requisição de Envio do *Buffer* 0 49
3.5.7 Instrução de Requisição de Envio do *Buffer* 1 50
3.5.8 Instrução de Requisição de Envio do *Buffer* 2 51
3.6 Mensagens CAN 52

4 Resultados Obtidos **53**

5 Conclusões **57**

6 ANEXO A - TMS320LF2407A **61**
6.1 Introdução 61
6.2 Características do TMS320LF2407A 61
6.3 Arquitetura do Dispositivo 64
6.4 Memória 65
6.4.1 Memória de Programa 65
6.4.2 Memória de Dados 66
6.4.3 Espaço I/O 68
6.5 Clocks e Modos de Baixa Potência 68
6.5.1 Pinos Associados com o Clock 68
6.5.2 Phase Locked Loop - PLL 69
6.5.3 Watchdog timer Clock - WDCLK 69
6.5.4 Modos de Baixa Potência 69
6.6 Entradas e Saídas Digitais - I/O 70

6.7 Gerenciador de Eventos 70
6.8 Conversor Analógico Digital 74
6.9 Interface de Comunicação Serial - SCI 75
6.10 Interface Periférica Serial (SPI) 77
6.11 Módulo CAN 79
6.12 Watchdog Timer 80

7 ANEXO B - ADuC812 **82**
7.1 Descrição Geral 82
7.2 Organização da Memória 83
7.3 Portas de I/O 85
7.4 *Interface* Serial UART 85
7.5 *Interface* SPI 87
7.5.1 Pino MISO 88
7.5.2 Pino MOSI 88
7.5.3 Pino SCLOCK 88
7.5.4 Pino SS 88
7.5.5 *Interface* SPI - Configuração Mestre 88
7.5.6 *Interface* SPI - Configuração Escravo 89
7.5.7 Modos de Operação 89
7.6 Módulos A/D e D/A 90
7.7 Temporizadores 91

8 ANEXO C - Controlador CAN MCP2510 **92**
8.1 Introdução 92
8.2 Transmissão de Mensagens 94
8.2.1 Prioridade de Transmissão 94
8.2.2 Iniciando a Transmissão 94
8.2.3 Pinos TXnRTS 95
8.2.4 Abortando uma Transmissão 95
8.3 Recepção de Mensagens 95
8.3.1 Prioridade de Recepção 96
8.3.2 Filtros e Máscaras de Aceitação 96
8.4 Pinos RX0BF E RX1BF 96
8.5 Temporização dos *Bits* 96
8.6 Detecção de Erro 98
8.7 Interrupções 99
8.8 Modos de Operação 100
8.9 Interface SPI 101

8.9.1 Instrução de Leitura 101
8.9.2 Instrução de Escrita 102
8.9.3 Instrução de Requisição de Transmissão 102
8.9.4 Instrução de Leitura de *Status* 103
8.9.5 Instrução de Modificação de *Bit* 104
8.9.6 Instrução de *Reset* 105

9 ANEXO D - Transceiver CAN MCP2551 106
9.1 Introdução 106
9.1.1 Modos de Operação 107

Lista de Figuras

2.1 Diagrama das Camadas do Modelo OSI Utilizadas pelo Protocolo CAN. 4
2.2 Capacidade Multi-Mestre. 5
2.3 Esquema da Comunicação Broadcast. 5
2.4 Taxas de Transmissão Especificadas pela Cia. 6
2.5 Diagrama de Blocos de um Nó CAN 8
2.6 Processo de Transmissão e Recepção de Mensagens 10
2.7 Esquema do Processo de Arbitração. 11
2.8 Diagrama de uma Mensagem CAN com Formato Padrão. 12
2.9 Diagrama de uma Mensagem CAN com Formato Estendido. 12
2.10 Campos de uma Mensagem de Dados. 13
2.11 *Bits* dos Campos de uma Mensagem de Dados com Formato Padrão. 14
2.12 *Bits* dos Campos de uma Mensagem de Dados com Formato Estendido. 14
2.13 Campos de uma Mensagem de Requisição de Dados. 15
2.14 Funcionamento do Processo de Requisição de Dados. 16
2.15 Campos de uma Mensagem de Erro. 16
2.16 Esquema do Processo de Sinalização de Erros. 17
2.17 Pacote de Sobrecarga. 18
2.18 Diagrama do Intervalo *Interframe* para Nós não Passivos. 19
2.19 Diagrama do Intervalo *Interframe* para Nós Passivos. 20
2.20 Esquema do Processo de Inserção de *Bits*. 20
2.21 Diagrama com os Campos de uma Mensagem que estão Submetidos ao Método da Inserção de *Bits*. 22
2.22 Diagrama com os Campos de uma Mensagem de Dados que são Submetidos ao Mecanismo de Verificação de *Bit*. 22
2.23 Polinomial Geradora da Seqüência CRC. 23
2.24 Diagrama com os Campos de uma Mensagem que estão Submetidos ao Cálculo do CRC. 23
2.25 Diagrama com os Campos de uma Mensagem que estão Submetidos ao Método de Detecção de Formato. 24
2.26 Diagrama com os Possíveis Estados de Erro de um Nó CAN. 25

2.27 Diagrama com os Segmentos que Formam um *Bit Time*. 26
2.28 Esquema com o Formato de um *Time Quanta*. 26
2.29 Esquema com o Processo de Sincronização com Transmissor Lento. 29
2.30 Esquema com o Processo de Sincronização com Transmissor Rápido. 29
2.31 Esquema do Barramento Diferencial. 30
2.32 Técnica de Sinalização Diferencial. 30
2.33 Processo de Imunidade as Interferências Eletromagnéticas. 31
2.34 Esquema de Funcionamento do *Transceiver*. 31
2.35 Diagrama de Blocos de um Módulo CAN *Stand-Alone*. 33
2.36 Diagrama de Blocos de um Módulo CAN Integrado. 33
2.37 Diagrama de Blocos de um Módulo CAN *Single-Chip*. 34
2.38 Esquema de um Filtro Simples. 35
2.39 Esquema de um Filtro Múltiplo. 35

3.1 Diagrama de Blocos do Sistema Implementado. 37
3.2 Diagrama com as possíveis transmissões e recepções de mensagens CAN do Sistema Implementado. 38
3.3 Tela do *Software Keil uVision*. 39
3.4 Tela do Aplicativo WSD. 40
3.5 Tela do *Software Code Composer*. 41
3.6 Diagrama Elétrico Simplificado do Nó CAN_1. 42
3.7 Diagrama Elétrico Simplificado do Nó CAN_3. 43
3.8 Tela com os Sinais dos Pinos SCLK e MOSI em uma Instrução de Leitura. 44
3.9 Tela com os Sinais dos Pinos MISO e CS em uma Instrução de Leitura. 44
3.10 Tela com os Sinais dos Pinos SCLK e MOSI em uma Instrução de Escrita. 45
3.11 Tela com os Sinais dos Pinos MISO e CS em uma Instrução de Escrita. 45
3.12 Tela com os Sinais dos Pinos SCLK e MOSI em uma Instrução de Leitura de *Status*. 46
3.13 Tela com os Sinais dos Pinos MISO e CS em uma Instrução de Leitura de *Status*. . . 46
3.14 Tela com os Sinais dos Pinos SCLK e MOSI em uma Instrução de Modificação de *Bit*. 47
3.15 Tela com os Sinais dos Pinos MISO e CS em uma Instrução de Modificação de *Bit*. . 47
3.16 Tela com os Sinais dos Pinos SCLK e MOSI em uma Instrução de *Reset*. 48
3.17 Tela com os Sinais dos Pinos MISO e CS em uma Instrução de *Reset*. 48
3.18 Tela com os Sinais dos Pinos SCLK e MOSI em uma Instrução de Requisição de Envio do *Buffer* 0. 49
3.19 Tela com os Sinais dos Pinos MISO e CS em uma Instrução de Requisição de Envio do *Buffer* 0. 49
3.20 Tela com os Sinais dos Pinos SCLK e MOSI em uma Instrução de Requisição de Envio do *Buffer* 1. 50

3.21 Tela com os Sinais dos Pinos MISO e CS em uma Instrução de Requisição de Envio do *Buffer* 0. 50
3.22 Tela com os Sinais dos Pinos SCLK e MOSI em uma Instrução de Requisição de Envio do *Buffer* 2. 51
3.23 Tela com os Sinais dos Pinos MISO e CS em uma Instrução de Requisição de Envio do *Buffer* 2. 51
3.24 Tela com as Informações de um Pacote de Dados e o *Bit* de Reconhecimento. 52

4.1 Tela do *Hyper Terminal* do PC no Nó CAN1 com as Informações do Sensor ST1 e do Potenciômetro P1. 53
4.2 Tela do *Hyper Terminal* do PC no Nó CAN1 com as Informações do Sensor ST1, do Potenciômetro P1 e do *Cooler*. 54
4.3 Tela do *Hyper Terminal* do PC no Nó CAN2 com as Informações do Sensor ST2 e do Potenciômetro P2. 54
4.4 Tela do *Hyper Terminal* do PC no Nó CAN2 com as Informações do Sensores ST1 e ST2 e dos Potenciômetros P1 e P2. 55
4.5 Tela do Osciloscópio com os Sinais PWM1 e PWM 2 com diferentes ciclos de trabalho. 56

6.1 Diagrama de Blocos do TMS320LF2407A. 63
6.2 Diagrama de Blocos da Arquitetura do TMS320LF2407A. 64
6.3 Esquema do Mapa da Memória de Programa. 66
6.4 Esquema do Mapa da Memória de Dados. 67
6.5 Esquema do Mapa com os Endereços do Espaço de I/O. 68
6.6 Diagrama de Blocos do Módulo EVA. 71
6.7 Diagrama de Blocos do Módulo EVB. 72
6.8 Diagrama de Blocos do Módulo SCI. 76
6.9 Diagrama de Blocos do Módulo SPI. 78
6.10 Diagrama de Blocos do Módulo CAN do TMS320LF2407A. 80
6.11 Diagrama de Blocos do Módulo WD. 81

7.1 Diagrama de Blocos do Microconversor ADuC812. 83
7.2 Diagrama com o Mapa do Espaço da Memória de Programa. 83
7.3 Diagrama com o Mapa do Espaço da Memória de Dados. 84
7.4 Diagrama de Blocos do Modelo de Programação do ADuC812 Via a Área SFR. . . . 85
7.5 Processo de Transmissão de *Bits* Utilizando o Modo 1. 87
7.6 Esquema de Conecção dos Pinos entre um Dispositivo Mestre e um Escravo. 87
7.7 Esquema dos Sinais nos Pinos da SPI nos Vários Modos de Operação. 89
7.8 Formato do Sinal Resultante Convertido no ADC. 90

8.1 Esquema da Pinagem do MCP2510 93

8.2 Diagrama de Blocos do MCP2510 93
8.3 Diagrama com os Sinais dos Pinos da *Interface* SPI em uma Instrução de Leitura. . . 101
8.4 Diagrama com os Sinais dos Pinos da *Interface* SPI em uma Instrução de Escrita. . . 102
8.5 Diagrama com os Sinais dos Pinos da *Interface* SPI em uma Instrução de Requisição de Transmissão. 103
8.6 Diagrama com os Sinais dos Pinos da *Interface* SPI em uma Instrução de Leitura de *Status*. 103
8.7 Exemplo de Funcionamento da Máscara em uma Instrução de Modificação de *Bit*. . . 104
8.8 Diagrama com os Sinais dos Pinos da *Interface* SPI em uma Instrução de Modificação de *bit*. 105
8.9 Diagrama com os Sinais dos Pinos da *Interface* SPI em uma Instrução de *Reset*. . . . 105

9.1 Esquema da Pinagem do MCP2551. 106
9.2 Diagrama de Blocos do MCP2551. 107

Lista de Tabelas

6.1 Descrição dos pinos utilizados pelos módulos EVA e EVB. 73

Capítulo 1

Introdução

Devido aos avanços tecnológicos, os sistemas se tornaram maiores e mais complexos. A necessidade de descentralizar a comunicação entre os dispositivos inteligentes de um sistema, adotando uma comunicação em rede, se tornou cada vez maior. A escolha por uma solução em rede permite um conjunto de benefícios em relação a uma solução centralizada, tais como [1]:

- Menos fios no sistema, o que torna a cablagem mais simples e barata;
- Ligações curtas a sensores analógicos sensíveis a ruído elétrico, antes dos sinais serem convertidos em mensagens digitais "menos susceptíveis"a ruído;
- Sistema flexível: unidades funcionais podem ser adicionadas ou removidas de forma simples;
- Manutenção do sistema: a eletrônica de várias unidades pode ser idêntica, permitindo troca entre elas;
- Cada unidade pode ser desenvolvida e testada individualmente de acordo com os requisitos exigidos pelo sistema.

Inicialmente, CAN [2] [3] [4] foi projetada para simplificar os complexos sistemas de fios existentes nos automóveis. Ao longo dos anos, CAN passou a ser utilizada também em outras áreas, e devido as suas características, podemos utilizá-la em sistemas que necessitam de controle distribuído em tempo real. Sistemas distribuídos são constituídos de diversos componentes de *hardware* e *software* que podem falhar, levando a um comportamento não esperado do sistema, ou mesmo, à não disponibilidade do serviço ofertado. Sistemas em tempo real, são sistemas que além dos requisitos lógicos estão submetidos a requisitos de natureza temporal [5]. Acessar informações e controlar processos distribuídos em tempo real torna-se essencial para a redução de custos e agilidade na tomada

de decisões. Assim, implementar sistemas que forneçam essas funcionalidades em *hardwares* e *softwares* abertos, com conectividade, é um grande desafio [6].

Diversos trabalhos utilizando a rede CAN têm sido desenvolvidos visando à monitoração e o controle de sistemas. Podemos encontrar aplicações na robótica [7]), no setor de transmissão de energia elétrica [6], no controle de telescópios [8], de máquinas [1], de helicópteros não tripulados [9], no controle remoto de residências [10] e outros.

Nesse projeto implementamos três nós CAN para implementar o controle de um processo que é constituido por: um subsistema de monitoração de temperatura; um subsistema de controle para acionamento de um sistema de resfriamento (um *cooler*); e um subsistema de controle do ciclo de trabalho de dois sinais PWM.

Capítulo 2

CAN

2.1 Introdução

A rede CAN surgiu de modo a satisfazer a crescente necessidade de segurança, conforto e comodidades existentes no mercado automotivo, associada às exigências governamentais no que diz respeito à diminuição de poluição, consumo e aumento da segurança, para isto a indústria tem desenvolvido muitos sistemas eletrônicos.

Com o aumento de complexidade dos sistemas eletrônicos presentes nos automóveis, a comunicação entre eles implica num aumento do número de fios e linhas de controle dedicadas, ou seja, à medida que vai se aumentando a funcionalidade dentro dos automóveis, a quantidade de fios aumenta e com ela, os custos os problemas de confiabilidade, a necessidade de reparos e diagnósticos de faltas durante o uso e a fabricação. Logo a necessidade de descentralizar a comunicação entre os dispositivos inteligentes desses sistemas, adotando uma comunicação em rede, se tornou cada vez maior.

Uma solução para esse problema foi encontrada por volta de meados dos anos 80, pelas companhias Robert Bosch, a rede CAN (*Controller Area Network*) [2]. A Bosch desenvolveu o protocolo, que se tornou um padrão internacional e é documentada na ISO11898 (para aplicações em alta velocidade) e ISO11519 (para aplicações em baixa velocidade). Muitos protocolos são descritos usando as 7 camadas do modelo OSI (*Open System Interconnection*). O protocolo CAN utiliza apenas a camada de enlace de dados (LLC e MAC) e uma das três partes da camada física, a subcamada de sinalização física (PLS), deste modelo. O restante da camada física (PMA e MDI) e as outras camadas não são definidos pela especificação CAN, mas elas podem ser definidas pelo projetista do sistema [4]. Na figura 2.1 é ilustrado o modelo OSI e as camadas utilizadas pelo protocolo CAN.

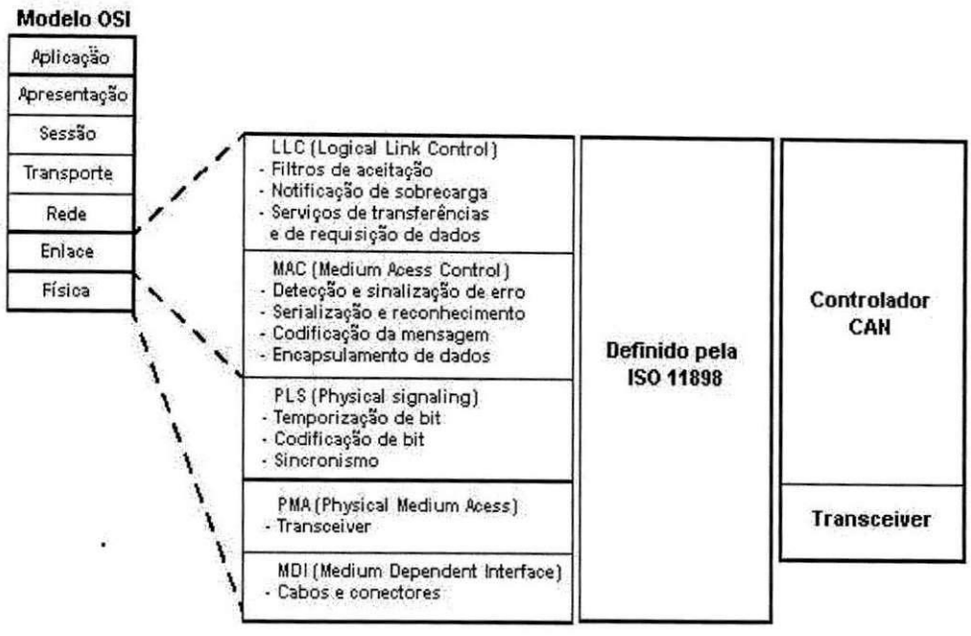

Figura 2.1: Diagrama das Camadas do Modelo OSI Utilizadas pelo Protocolo CAN.

2.2 Aplicações

Inicialmente esse barramento foi desenvolvido para o uso em automóveis, mais atualmente é encontrado em muitos campos de aplicação, dentre as principais citamos:

- Veículos (marítimos, áereos e terrestres) - carros de passeio, *off-road*, trens, sistemas de semáforo (trens e carros), eletrônica marítima, máquinas agrícolas, helicópteros, transporte público;
- Sistemas de controle industrial - controle de planta, de maquinário, de robôs, sistemas de supervisão;
- Automação predial - controle de elevadores, de ar condicionado, de iluminação;
- Aplicações específicas - sistemas médicos, telescópios, simuladores de vôo, satélites artificiais e outros.

Em geral, CAN é disponível em todas as aplicações onde vários subsistemas e dispositivos inteligentes precisam se comunicar.

2.3 Características

CAN é um protocolo de comunicação serial que é usado eficientemente para aplicações de controle distribuído em tempo real com alto nível de segurança. É um sistema em barramento com capacidades multi-mestre, isto é, vários nós podem pedir acesso ao meio de transmissão em simultâneo. Na figura 2.2 é ilustrado esse processo.

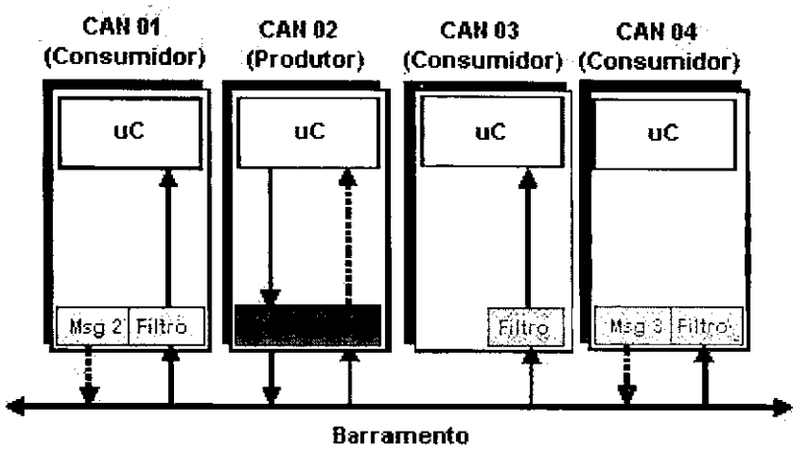

Figura 2.2: Capacidade Multi-Mestre.

Este protocolo suporta também a comunicação *broadcast*, isto é, permite que uma mensagem seja transmitida a um conjunto de receptores simultaneamente. Na figura 2.3 é ilustrada essa característica.

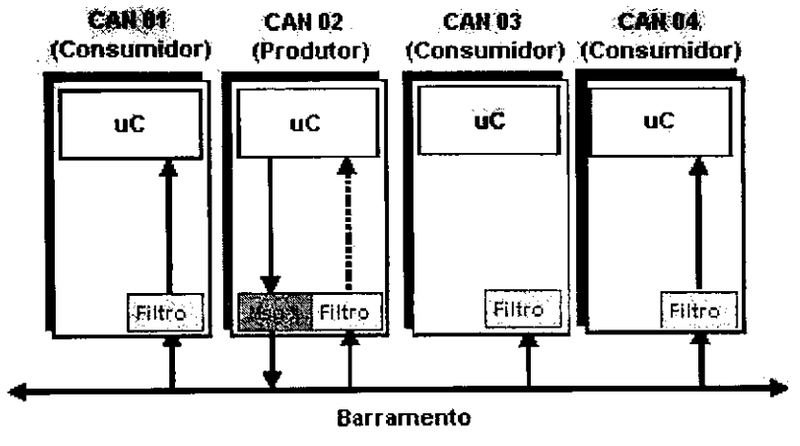

Figura 2.3: Esquema da Comunicação Broadcast.

Nas redes CAN não existe um endereço do destinatário, em vez disso são transmitidas mensagens que possuem um determinado identificador. Assim, um nó envia uma mensagem a todos os outros nós CAN e cada um deles decide, baseado no identificador recebido, se deve ou não processar a mensagem. O identificador determina também a prioridade intrínseca da mensagem, ao competir com outras pelo acesso ao barramento.

A informação transmitida possui um tamanho curto. Cada mensagem CAN pode conter um máximo de 8 *bytes* de informação útil, sendo no entanto possível transmitir blocos maiores de dados recorrendo a segmentação.

A taxa de transmissão depende do comprimento do barramento, pois o tempo de transmissão e recepção de um *bit* nunca deve ser inferior ao dobro de atraso de propagação no barramento. A taxa máxima especificada é de 1 Mbps, correspondendo este valor a sistemas com barramento de comprimento de até 30 m. Para distâncias superiores a taxa de transmissão, recomendada, diminui. Se a distância do barramento for superior a 1 Km pode ser necessária a utilização de dispositivos repetidores ou pontes. Os repetidores têm a função de reforçar o sinal elétrico para que esse possa percorrer maiores distâncias. Alguns dos valores recomendados pela Cia (*Can In Automation*), são ilustrados na figura 2.4.

Taxas de Transmissão	Comprimento do Barramento	Tempo de Bit
1 Mbit/s	30 m	1 uS
800 Kbit/s	50 m	1,25 uS
500 Kbit/s	100 m	2 uS
250 Kbit/s	250 m	4 uS
125 Kbit/s	500 m	8 uS
62,5 Kbit/s	1000 m	20 uS
20 Kbit/s	2500 m	50 uS
10 Kbit/s	5000 m	100 uS

Figura 2.4: Taxas de Transmissão Especificadas pela Cia.

O número de elementos num sistema CAN está, teoricamente, limitado pelo número possível de identificadores diferentes. Este número limite é significativamente reduzido por limitações físicas do *hardware*. Existem no mercado de integrados *transceivers* que permitem ligar pelo menos 110 nós.

O CAN permite flexibilidade uma vez que podem ser adicionados novos nós a uma rede CAN sem requerer alterações do *software* ou *hardware* dos restantes nós, se o novo nó não for emissor, ou se o novo nó não necessitar da transmissão de dados adicionais. Mesmo sendo um nó que transmita mensagens, as modificações nos demais nós do sistema serão mínimas.

Uma característica interessante, é o fato do controlador CAN de cada estação registrar os erros, e os avaliar estatisticamente, de modo a desencadear ações com eles relacionadas. Essas ações podem corresponder ao desligamento, ou não, da estação que provocou os erros. Isto torna o protocolo bastante eficaz em ambientes ruidosos.

Em resumo, o grande interesse pelo CAN ocorre devido às suas diversas características, tais como:

- Padrão ISO/OSI;
- Capacidade multi-mestre;
- Capacidade *broadcast*;
- Atribuição de prioridade às mensagens;
- Simplicidade;
- Flexibilidade de configuração;
- Elevadas taxas de transferência (1Mbps);
- Considerável imunidade ao ruído;
- Distinção entre erros temporários e erros permanentes dos nós;
- Capacidade eficaz de detectar e sinalizar erros;
- Desligamento de nós com defeitos;

- Requisição de mensagens;
- Retransmissão automática de mensagens "em espera"logo que o barramento esteja livre;
- Reduzido tempo de latência (tempo real);
- Redução do número de fios a serem utilizados;
- Baixo preço;
- *Hardware* padrão.

2.4 Implementação de um Nó CAN

Para implementar um nó CAN, é necessário ter, segundo a CiA, três componentes básicos: um *transceiver*, um controlador CAN e um microcontrolador. Na figura 2.5 é ilustrado um nó CAN.

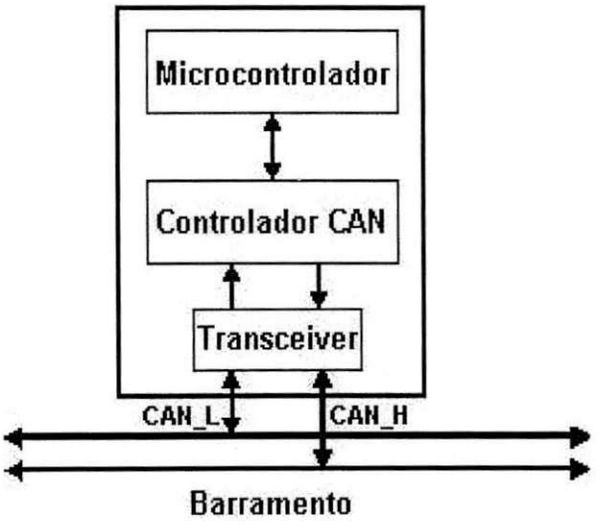

Figura 2.5: Diagrama de Blocos de um Nó CAN

2.5 Método de Endereçamento

Quando são transmitidos dados utilizando o CAN, não existe o endereço de destino da mensagem. Existem identificadores nas mensagens que servem para caracterizar o conteúdo da mensagem. Portanto, ao receber uma mensagem é de competência de cada nó da rede, decidir se a mensagem é ou não válida para esse nó, realizando para isso um teste de aceitação do identificador da mesma. Este teste é designado por filtragem. Existem vários dispositivos controladores que permitem diferentes níveis de sofisticação desta filtragem. O identificador estabelece também a prioridade da mensagem. Isto é importante para a atribuição do barramento quando vários nós estão competindo pelo acesso ao meio de transmissão.

Os nós de um barramento CAN estão conectados logicamente por uma função *wired-AND* ou *wired-OR*. Isto significa que podemos distinguir níveis dominantes e recessivos no barramento. Em uma conecção *wired-AND*, como implementado pela Intel, Motorola, Siemens e outros, o nível lógico '1' é um valor de *bit* recessivo e o nível lógico '0' é um valor de *bit* dominante [11]. Deste modo, na ocorrência de duas escritas simultâneas no barramento, uma escrita dominante (0) e outra recessiva (1), o valor que predominará no barramento será o dominante. A partir de agora, nos referiremos sempre à lógica *wired-AND*.

2.6 Processo de Transmissão e Recepção de Mensagens CAN

Se a unidade central de processamento (CPU), de um dos nós desejar enviar uma mensagem para um ou mais nós da rede, este "passa"os dados a serem transmitidos e o respectivo identificador para o controlador CAN ("Preparar") desse nó utilizando algum tipo de *interface*. Isto é tudo o que o CPU necessita realizar para iniciar a transferência de informação. A mensagem é então composta, e transmitida pelo controlador CAN, assim que o mesmo conseguir acesso ao barramento ("Enviar mensagem"). Todos os outros nós da rede CAN tornam-se nesse momento receptores dessa mensagem ("Receber mensagem"). Cada nó da rede tendo recebido corretamente a mensagem, enviará um *bit* de reconhecimento e realizará um teste de aceitação para determinar se os dados recebidos são ou não relevantes para essa estação ("Seleção"). Se os dados tiverem significado são processados ("Aceite"), caso contrário, são rejeitados ("Não Aceite"). O processo de transmissão e recepção de mensagens CAN está ilustrado na figura 2.6.

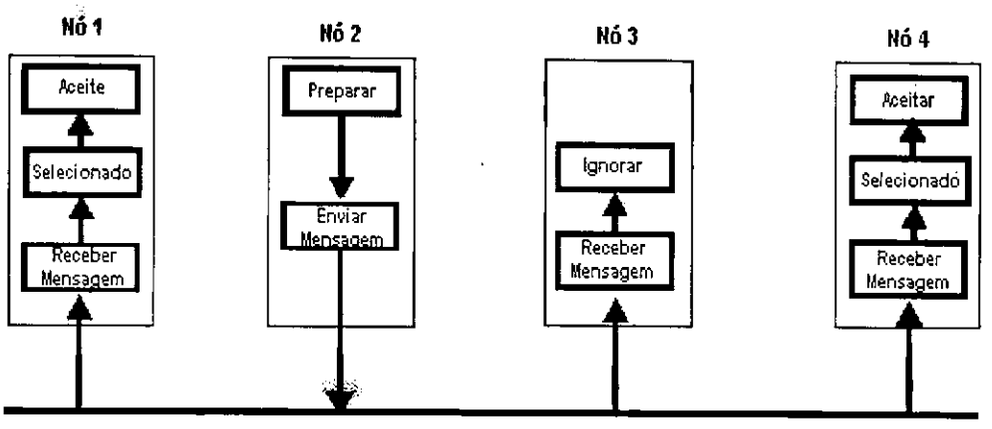

Figura 2.6: Processo de Transmissão e Recepção de Mensagens

2.7 Processo de Arbitragem Não-Destrutivo

Para que os dados sejam processados em tempo real, estes devem ser transferidos rapidamente. Isto exige um meio físico que permita uma elevada taxa de transmissão e chamadas rápidas à alocação do barramento quando vários nós tentam transmitir simultaneamente. No processamento em tempo real a urgência da troca de mensagens pela rede pode ser significativamente diferente: uma grandeza que varie rapidamente deve ser transmitida com maior freqüência e menores atrasos do que outras que variem menos. No CAN, a prioridade com que uma mensagem é transmitida relativamente à outra é especificada pelo identificador da respectiva mensagem, como dito na seção 1.5. A prioridade das mensagens é definida durante a fase de projeto do sistema sob a forma de valores binários. As mensagens com os identificadores que possuem menor valor, terão a maior prioridade de acesso ao barramento. O CAN é uma rede CSMA/CD + AMP (*Carrier Sense Multi-Acess with collision Detection*), isto quer dizer que os nós atrasam a transmissão se o barramento estiver ocupado, mas quando a condição de barramento livre for detectada, qualquer nó pode iniciar a transmissão. Os conflitos de acesso ao barramento são solucionados por comparação orientada ao *bit* (*bitwise*) dos identificadores das mensagens e funciona da seguinte forma. Enquanto transmite o identificador da mensagem, cada nó monitorará o barramento, se o *bit* transmitido for "recessivo", nível lógico '1', e for monitorizado um *bit* "dominante", nível lógico '0', o nó desiste da transmissão e inicia a recepção dos dados que chegam. O nó que transmite a mensagem com o menor identificador, ganha acesso ao barramento e continua a transmissão. Na figura 2.7 é ilustrado como ocorre a arbitração. Até o *bit* 6

todos os nós enviaram o mesmo valor. Porém, no *bit* 5 o nó 2 transmitiu um *bit* recessivo, perdendo assim a disputa para o nó 1 e 3. No *bit* 2 o nó 1 enviou um *bit* recessivo e também perdeu a disputa pelo acesso ao meio de transmissão. Desta forma o nó 3 ganhou a disputa pelo barramento e seguiu transmitindo a mensagem, os demais passaram a ser apenas receptores dessa mensagem.

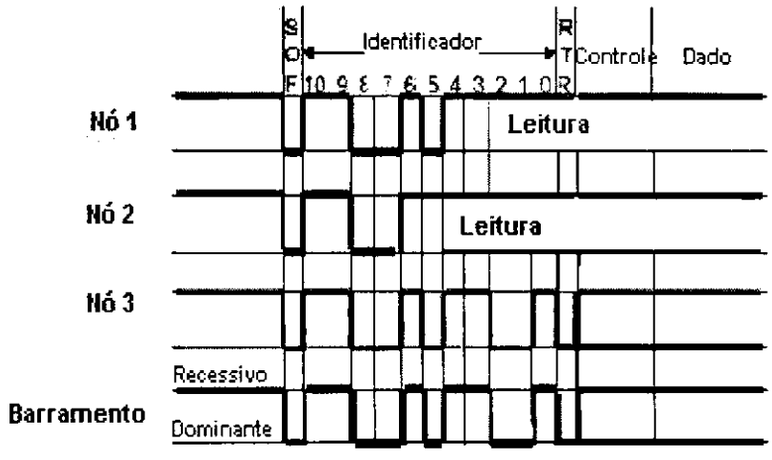

Figura 2.7: Esquema do Processo de Arbitração.

Isto significa que a arbitragem é não-destrutiva, uma vez que a transmissão da mensagem de menor identificador não sofreu atraso.

2.8 Formato das Mensagens CAN

Existem dois formatos que são utilizados, o CAN Padrão (versão 2.0A) e o CAN Estendido (versão 2.0B) [2]. A principal diferença entre esses dois formatos, é quanto ao número de *bits* dos identificadores. No formato padrão, as mensagens possuem identificadores com 11 *bits*. No formato estendido, possuem identificadores com 29 *bits*. Nas Figuras 2.8 e 2.9 estão ilustradas, respectivamente, diagramas das mensagens com os formatos padrão e estendido.

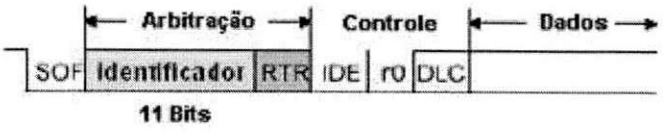

Figura 2.8: Diagrama de uma Mensagem CAN com Formato Padrão.

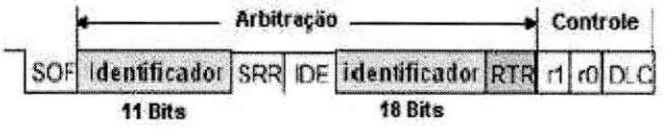

Figura 2.9: Diagrama de uma Mensagem CAN com Formato Estendido.

Existem duas versões de controladores, o CAN1.0 e o CAN2.0. O CAN2.0 é completamente compatível com o CAN1.0. A versão 2.0 possui duas variações, o 2.0A (Padrão) e o 2.0B (Estendido). Tanto na versão 1.0 como na 2.0A, os identificadores possuem 11 *bits* de comprimento. Na versão 2.0B, podemos ter tanto identificadores de 11 *bits*, como identificadores de 29 *bits*. A maioria dos novos controladores respeita a versão 2.0. Existem regras de compatibilidade que devem ser consideradas para transmitir e receber os dois tipos de formatos das mensagens. De forma a manter a compatibilidade com as versões anteriores, o 2.0B pode ser um controlador ativo ou passivo:

- Controladores CAN 2.0B ativos transmitem e recebem ambos os formatos de mensagens estendidas ou padrão;
- Controladores CAN 2.0B passivos transmitem e recebem o formato de mensagens padrão e ignoram sem erros as mensagens estendidas;
- Controladores CAN 1.0 geram erros quando detectam mensagens estendidas.

2.9 Tipos de Mensagens

A transmissão e recepção de informação num sistema CAN são efetuadas e controladas através de quatro tipos diferentes de mensagens (Pacotes), são elas:

- Pacote de dados;
- Pacote de requisição de dados;
- Pacote de erro;
- Pacote de sobrecarga.

Os dois primeiros tipos de mensagens são utilizados durante a operação normal da rede CAN: a mensagem de dados é utilizada para transferir dados de um nó para outro, e a mensagem de requisição de dados é usada para requisitar dados de um nó. Esses dois tipos de mensagens iniciam sempre com o um *bit* dominante SOF (*Start Of Frame*) e terminam com 7 *bits* recessivos EOF (*End Of Frame*). Os outros tipos de mensagens são usados para sinalizar um estado anormal da rede CAN: a mensagem de erro sinaliza a existência de um estado de erro e a mensagem de sobrecarga sinaliza que um nó particular não está pronto para receber mensagens. A seguir, explicaremos com detalhes esses 4 tipos de mensagens.

2.9.1 Pacote de Dados

Essa mensagem contém os dados do nó emissor para o nó receptor. Nas figuras 2.10, 2.11 e 2.11 são ilustradas, respectivamente, diagramas dos campos e os *bits* dos respectivos campos de uma mensagem de dados com formato padrão e estendido.

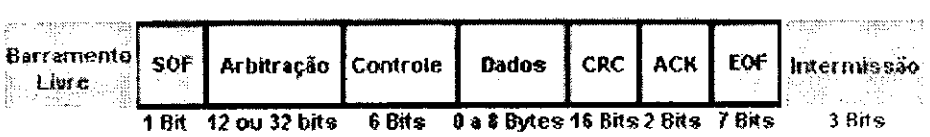

Figura 2.10: Campos de uma Mensagem de Dados.

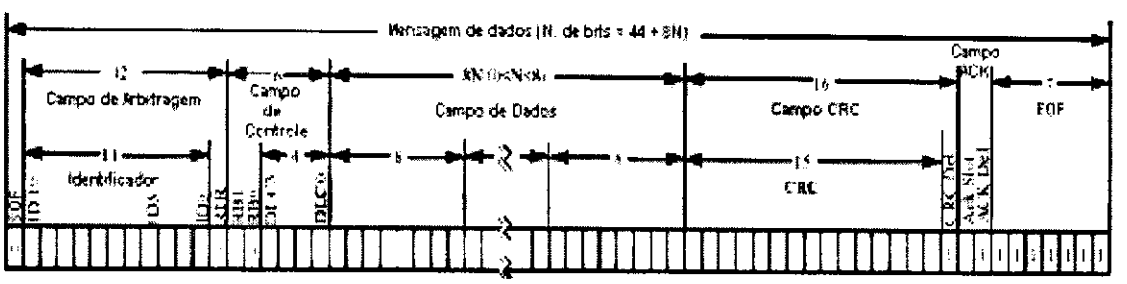

Figura 2.11: *Bits* dos Campos de uma Mensagem de Dados com Formato Padrão.

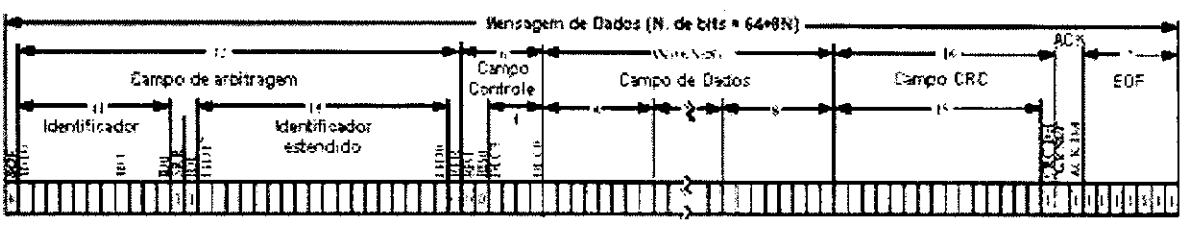

Figura 2.12: *Bits* dos Campos de uma Mensagem de Dados com Formato Estendido.

Veja que a mensagem é formada por campos, que possuem um determinado número de *bits*. A seguir, explicaremos cada um desses campos.

- **SOF (*Start Of Frame*)** - Marca o início de uma mensagem. Quando o barramento se encontra ocioso, uma borda de descida do SOF sincroniza os nós da rede;
- **Campo de Arbitração** - Este campo depende do formato do pacote. No formato padrão, é constituído pelo identificador e pelo *bit* RTR (*Remote Transmit Request*). Esse *bit* indica se a mensagem é de dados ou de requisição de dados. Caso ele tenha valor dominante, a mensagem será de dados. Caso ele tenha valor recessivo, a mensagem será de requisição de dados. No formato estendido, é constituído pelo identificador e pelos *bits* RTR, SRR (*Substitute Remote Request*) e IDE (*Identifier Extension*). O *bit* RTR tem o mesmo significado que no formato padrão. O *bit* SRR é um *bit* recessivo. Este *bit* é transmitido na mesma posição que o *bit* RTR ocupa em mensagens com o formato padrão. O *bit* IDE faz a distinção entre as mensagens de formato padrão e as mensagens de formato estendido. Em mensagens com formato padrão esse *bit* é "dominante", enquanto que em mensagens com formato estendido, esse *bit* é "recessivo".

- **Campo de Controle** - Este campo é composto por seis *bits*, sendo 2 *bits* reservados para futura expansão, seguidos pelo campo DLC (Data Length Code). Os *bits* reservados são transmitidos como "dominantes". O campo DLC possui 4 *bits* que indicam o número de *bytes* do campo de dados;
- **Campo de Dados** - Contém os dados, isto é, a informação útil da mensagem;
- **Campo CRC** - Esse campo contém 16 *bits*. Ele é composto pela seqüência CRC, que são 15 *bits*, e pelo delimitador CRC, que é um *bit* recessivo. Esse campo contém elementos que são usados para detecção de erros na mensagem. Pode identificar até 5 falhas. O cálculo da seqüência CRC é feito tendo em conta uma polinomial geradora;
- **Campo de Reconhecimento (ACK)** - Esse campo é constituído pelo *bit SLOT* ACK e pelo delimitador de ACK. Os transmissores enviam ambos os *bits* em recessivo. Os nós receptores ao receberem corretamente a seqüência CRC, enviam um *bit* dominante no *SLOT* ACK;
- **EOF (*End Of frame*)** - É composto por 7 *bits* recessivos. Delimita o fim da mensagem.

2.9.2 Pacote de Requisição de Dados

O pacote de requisição de dados é semelhante à mensagem de dados, existem apenas duas diferenças. O *bit* RTR é transmitido recessivo, e o campo de dados não existe. Na figura 2.13 estão ilustrados, os campos de uma pacote de requisição de dados.

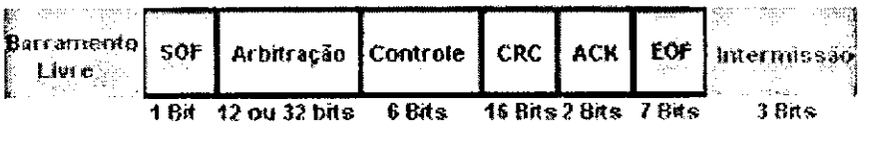

Figura 2.13: Campos de uma Mensagem de Requisição de Dados.

Um nó que deseja receber uma determinada informação pode requisitar esses dados a outro nó. Para isso, ele envia uma mensagem de requisição de dados, isto é, um pedido de dados. O identificador da mensagem de requisição de dados deverá ter o mesmo valor da mensagem de dados requisitada. Os *bits* DLC do campo de controle desse tipo de mensagem, indicarão o número de *bytes* que a mensagem requisitada deverá ter.

Na figura 2.14 é ilustrado o funcionamento de uma requisição de dados. O nó 1 envia uma requisição de dados com o identificador do mesmo, indicado pela marca (1) na figura. O nó que responde por esse dado é o nó 2. O nó 2, então envia o dado requisitado indicado pela marca (2) ao barramento e todos nós lêem este dado.

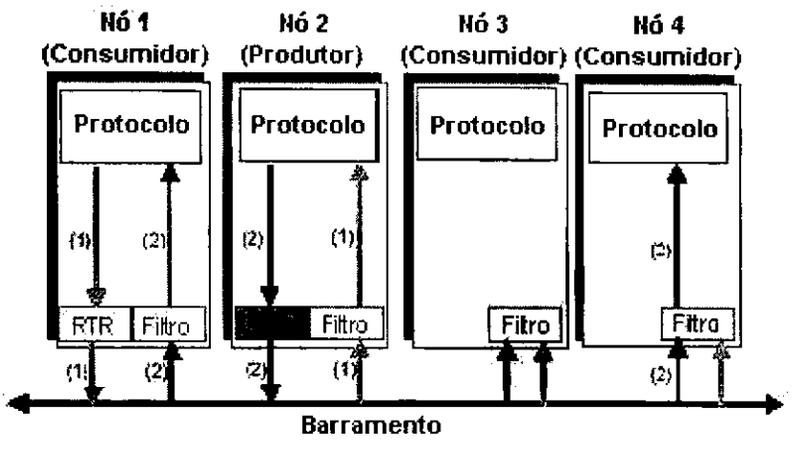

Figura 2.14: Funcionamento do Processo de Requisição de Dados.

2.9.3 Pacote de Erro

Esse pacote é transmitido por qualquer nó que detecte um erro. Sua função é notificar a ocorrência de falhas. Na figura 2.15 é ilustrado a forma do pacote de erro.

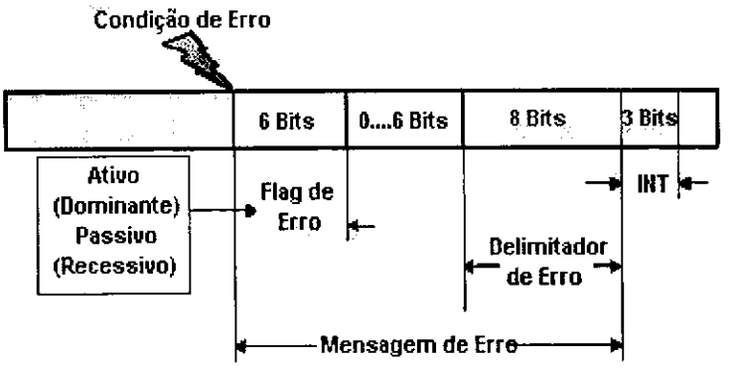

Figura 2.15: Campos de uma Mensagem de Erro.

Ele é formado pelos campos, *flag* de erro e delimitador de erro. A seguir veremos o significado de cada um desses dois campos.

- ***Flag* de erro** - É o campo que sinaliza a existência de um erro. Existem dois tipos de *flag* de erro: O ativo e o passivo. Um nó em estado de erro ativo envia um *flag* de erro ativo (dominante), enquanto um nó em estado de erro passivo envia um *flag* de erro passivo (recessivo). Posteriormente veremos a classificação de um nó ativo e de um nó passivo. Um campo de *flag* de erro é enviado pelo nó que detectou o erro, e se esse nó estiver em estado ativo, sobrescreverá o dado corrompido. Quando os outros nós da rede recebem a seqüência de 6 *bits* dominantes referente ao *flag* de erro, irá ocorrer uma violação de *bit stuffing* (método da inserção de *bits*) e todos os nós enviarão ao mesmo tempo um outro *flag* de erro. Se o nó que identificou o erro estiver atualmente sendo classificado como um nó passivo, ele enviará um *flag* de erro recessivo e mais o delimitador, também em recessivo. Portanto erros sinalizados por nós em estado de erro passivo não afetarão o barramento, isolando desta forma os nós que tenham uma taxa de erros grandes.
- **Delimitador de erro** - É transmitido pelo nó que continha o erro. Ele é formado por 8 *bits* recessivos. Sua função é recomeçar comunicações no barramento após uma falha.

Na figura 2.16 é ilustrado que o receptor 2 detecta um erro e torna-o público para os outros nós.

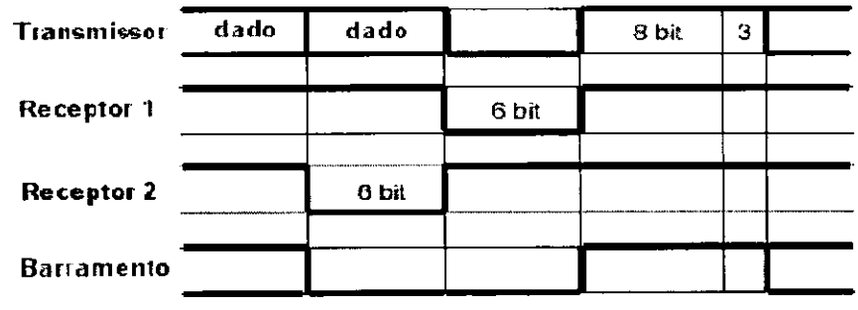

Figura 2.16: Esquema do Processo de Sinalização de Erros.

2.9.4 Pacote de Sobrecarga

Essa mensagem é utilizada para sinalizar a um transmissor que o receptor não está pronto para receber às próximas mensagens. O seu formato é semelhante à mensagem de erro, exceto que o flag de sobrecarga é sempre dominante e que o seu envio ocorre sempre depois do fim de uma mensagem de dados ou de requisição de dados, de modo a obter um atraso extra até a próxima mensagem de dados ou de requisição de dados. Na figura 2.17 é ilustrado esse tipo de mensagem.

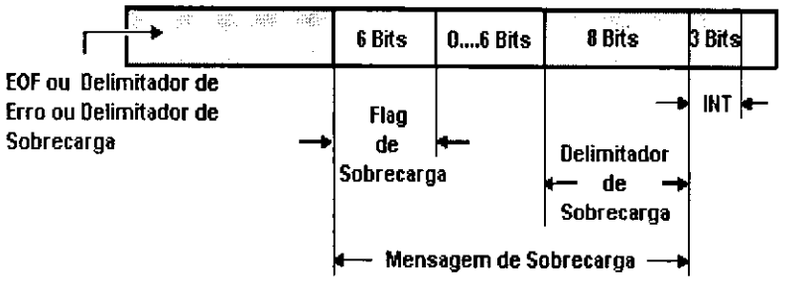

Figura 2.17: Pacote de Sobrecarga.

Veja que essa mensagem é formada pelo *flag* de sobrecarga e pelo delimitador de sobrecarga.

- ***Flag* de Sobrecarga** - Consiste em seis *bits* dominantes. Este *flag* destrói o formato fixo do campo de Intermissão, pelo fato de que todos os outros nós também detectam uma condição de sobrecarga e iniciam a transmissão de uma *flag* de sobrecarga. As condições que originam a transmissão de uma *flag* de sobrecarga são: condições internas de um receptor, que requer um atraso da próxima mensagem de dados ou de requisição de dados. Nestas condições a mensagem de sobrecarga só pode ter início durante o primeiro *bit* de uma intermissão; a detecção de um *bit* dominante durante a intermissão. Neste caso a mensagem de sobrecarga tem início depois de ser detectado o *bit* dominante. Após a transmissão de um *flag* de sobrecarga, qualquer nó poderá monitorar o barramento até detectar um *bit* recessivo.

- **Delimitador de Sobrecarga** - O delimitador de sobrecarga é composto por oito *bits* recessivos. Após a transmissão de uma *flag* de sobrecarga, todos os nós monitorizam o barramento até detectar uma transição de "dominante"para "recessivo". Nesse instante todos os nós terminaram o envio da sua *flag* de sobrecarga e todos iniciam simultaneamente a transmissão de mais sete *bits* "recessivos"em simultâneo, para completar o delimitador de sobrecarga com o comprimento de oito *bits*.

2.10 Intervalo *Interframe*

As mensagens de dados e de requisição de dados são separadas de mensagens precedentes independentemente do tipo destes (dados, remota, erro ou sobrecarga), por um campo de *bits* designado por intervalo interframe. As mensagens de erro e de sobrecarga não são precedidas por esse campo de *bits*. O intervalo interframe contém os seguintes campos:

Campo de intermissão - Esse campo consiste de 3 *bits*. Sua função é permitir que todos os controladores CAN se preparem para a próxima tarefa a ser executada. A única ação que pode ser feita é assinalar uma condição de sobrecarga;

- **Barramento Livre** - Esse campo possui tamanho arbitrário. A detecção de um *bit* dominante durante esse estado é interpretada como início de uma mensagem;
- **Suspensão de Transmissão** - É verificada quando um nó passivo ao erro, transmite 8 *bits* recessivos, após ter transmitido uma mensagem. Esse campo existe somente para os nós passivos ao erro que tenham sido emissoras de mensagens anteriores. Nas figuras 2.18 e 2.19 estão ilustradas, respectivamente, o intervalo interframe para nós que não sejam passivos ao erro ou que tenham sido receptoras da mensagem anterior e para nós passivos ao erro que tenham transmitido a mensagem anterior.

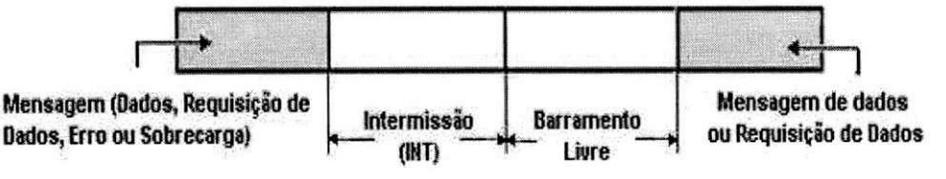

Figura 2.18: Diagrama do Intervalo *Interframe* para Nós não Passivos.

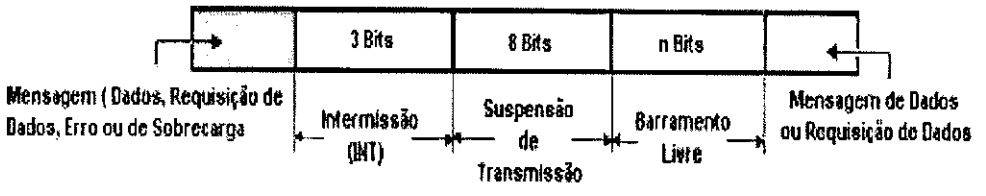

Figura 2.19: Diagrama do Intervalo *Interframe* para Nós Passivos.

2.11 Codificação

Os bits de uma mensagem CAN são codificados pelo método NRZ (*Non-Return-to-Zero*). Isto significa que durante o tempo de *bit*, um *bit* é classificado como recessivo ou dominante. Na codificação NRZ um grande número de *bits* com a mesma polaridade pode ser usado para resincronização do sistema. Então, utiliza-se o método de inserção de *bits* para sincronizar todos os nós do barramento. Isto significa que durante a transmissão de uma mensagem, um máximo de 5 *bits* podem ter a mesma polaridade. Na figura 2.20 é ilustrado o método da inserção de *bits* em uma transmissão de mensagem.

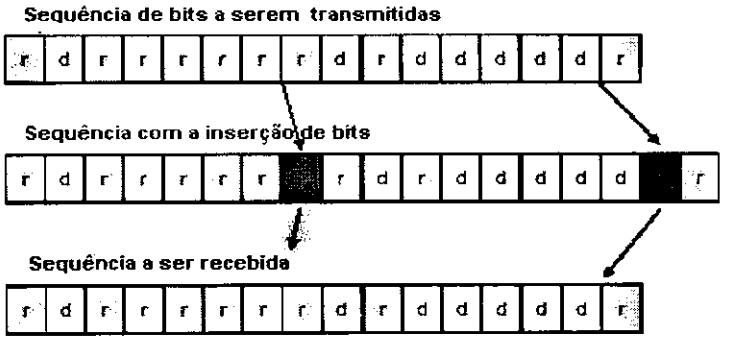

Figura 2.20: Esquema do Processo de Inserção de *Bits*.

Os seguintes campos das mensagens de dados e de requisição de dados (SOF, arbitração, controle, dados e CRC) são codificados pelo método da inserção de *bits*. Os demais campos e as mensagens de erro e de sobrecarga tem forma fixa e não são sujeitas a este método.

2.12 Métodos de Detecção e Sinalização de Erros

Para a detecção de erros, CAN implementa dois mecanismos ao nível de *bit* e três ao nível da mensagem. Esses mecanismos são:

- Verificação da regra de inserção de *bits*;
- Verificação dos *bits*;
- Verificação da seqüência CRC de 15 *bits*;
- Verificação de formato;
- Verificação do reconhecimento (ACK).

Esses tipos de mecanismos são usados para verificar a ocorrência de determinados tipos de erros. A seguir veremos esses tipos de erros, como eles são detectados utilizando esses mecanismos, e como são sinalizados.

2.12.1 Erro de Inserção de *Bits*

Se um nó detectar uma seqüência de mais de 5 *bits* com mesmo nível lógico, um erro de inserção de *bits* será detectado. Então, o nó que detectou esse erro transmitirá uma mensagem de erro no próximo *bit*. Nas mensagens de dados e de requisição de dados, essa técnica é usada para garantir a sincronização de todos os nós da rede e também aumentar a imunidade ao ruído. Existem alguns campos das mensagens de dados e de requisição de dados que não são afetados por esse mecanismo, como foi dito na seção 1.10. Na figura 2.21 são ilustrados os campos das mensagens de dados e de requisição de dados que estão submetidos ao método da inserção de *bits*.

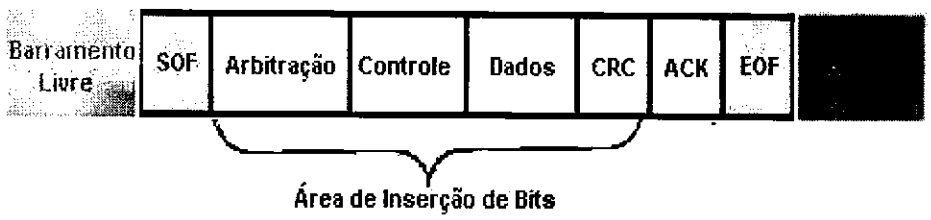

Figura 2.21: Diagrama com os Campos de uma Mensagem que estão Submetidos ao Método da Inserção de *Bits*.

2.12.2 Erro de *Bit*

Um nó que envia um *bit* ao barramento, também monitora o barramento. Quando o valor monitorado é diferente do valor enviado, pode-se dizer que um erro de *bit* ocorreu. Para sinalizar esse tipo de erro, uma mensagem de erro será transmitida ao barramento pelo nó que detectou o erro. Na figura 2.22são ilustrados os campos de uma mensagem de dados que são submetidos ao mecanismo de verificação de *bit*.

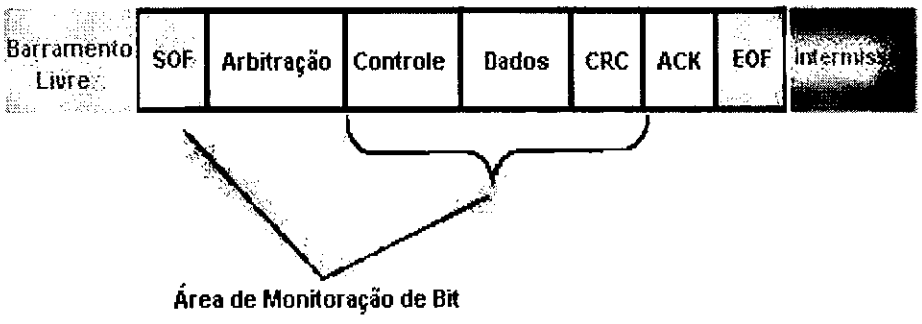

Figura 2.22: Diagrama com os Campos de uma Mensagem de Dados que são Submetidos ao Mecanismo de Verificação de *Bit*.

2.12.3 Erro de CRC

Ao receber uma mensagem de dados ou de requisição de dados, o nó receptor verifica se o CRC recebido é idêntico ao CRC calculado por ele. Caso sejam diferentes um erro de CRC ocorrerá. Neste caso, uma mensagem de erro será enviada ao barramento logo após o delimitador de ACK. O CRC é calculado desde o SOF até o fim do campo de dados. Nas figuras 2.23 e 2.24 são ilustrados, respectivamente, a equação polinomial utilizada para geração da seqüência CRC, e os campos de uma mensagem de dados que são submetidos ao mecanismo de verificação da seqüência CRC.

$$X^{15} + X^{14} + X^{10} + X^{8} + X^{7} + X^{4} + X^{3} + 1$$

Figura 2.23: Polinomial Geradora da Seqüência CRC.

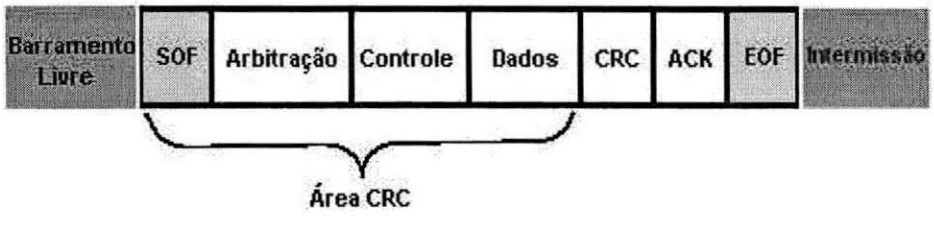

Figura 2.24: Diagrama com os Campos de uma Mensagem que estão Submetidos ao Cálculo do CRC.

2.12.4 Erro de Formato

Ocorre quando um *bit* de valor fixo (ACK *slot*, delimitador de CRC e EOF) contém um valor ilegal. Neste caso, uma mensagem de erro será gerada. Na figura 2.25 são ilustrados os *bits* de uma mensagem de dados que estão submetidos a esse método de detecção de erro.

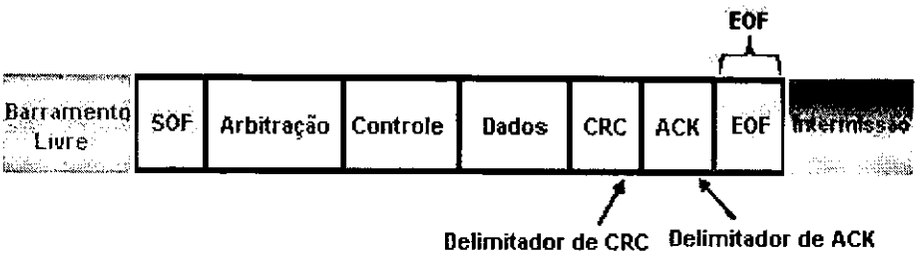

Figura 2.25: Diagrama com os Campos de uma Mensagem que estão Submetidos ao Método de Detecção de Formato.

2.12.5 Erro de Reconhecimento

Quando o transmissor envia uma mensagem, ele manda o ACK *slot* em recessivo, porém ao lê-lo o receptor mudará esse *bit* para dominante, e o envia de volta para o transmissor. Isso indica ao transmissor que ao menos um receptor recebeu a mensagem enviada. Se o transmissor não receber o ACK *slot* em dominante, significa que nenhum nó da rede leu o dado. Isso causará um envio de uma mensagem de erro para sinalizar essa falha no próximo *bit*, logo após o delimitador de ACK.

2.13 Estados de Erro

O CAN possui um algoritmo de confinamento de erros que é baseado em dois registradores, o REC (Contador de Erros de Recepção) e o TEC (Contador de Erros de Transmissão), que controlam o estado atual do nó. Existem três estados possíveis:

- **Erro ativo** - É o estado inicial dos nós. Neste estado, os nós enviam *flags* de erro ativo.
- **Erro passivo** - Quando existirem mais de 127 erros no REC ou no TEC, o estado passa de erro ativo para erro passivo. Neste estado os nós enviam, *flags* de erro passivo.
- **Desconectado** - Quando o TEC acusa mais de 255 erros o nó será desconectado do barramento. O nó só sairá desse estado, através de um *reset*.

Na figura 2.26 são ilustrados um diagrama com os possíveis estados de erro de um nó CAN.

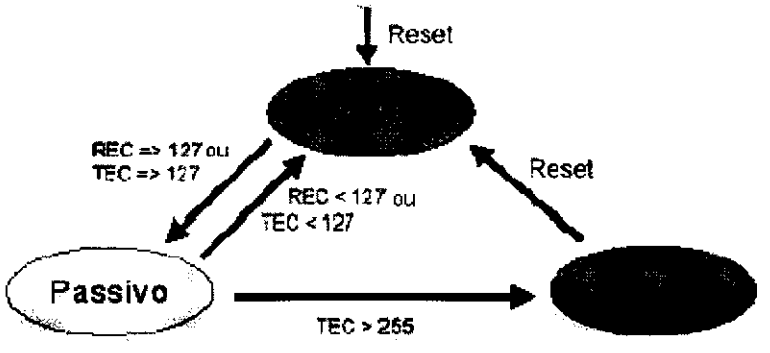

Figura 2.26: Diagrama com os Possíveis Estados de Erro de um Nó CAN.

2.14 *Bit Time*

O protocolo CAN permite programar a taxa de transmissão, o ponto de amostragem do *bit* e o número de vezes que o *bit* será amostrado. Com essas características a rede pode ser otimizada para uma dada aplicação.

Uma rede CAN consiste de vários nós, cada um com o seu *clock* individual. Por esse motivo podem ocorrer perdas de fase nos diferentes nós. Os nós CAN possuem um algoritmo de sincronização que compensa perdas de fase enquanto estão recebendo uma mensagem.

O *bit time* é o período de transmissão e recepção de um *bit* na ausência de resincronização. Todos os nós da rede devem ter um mesmo *bit time*. Ele é formado por quatro segmentos: segmento de sincronização (Seg_Sinc), segmento de propagação (Seg_Prop), segmento1 de fase (Seg_Fase1) e segmento 2 de fase (Seg_Fase2). Cada segmento do *bit time* duram um determinado tempo inteiro, chamado de *time quanta* (tq). O *time quanta* é uma unidade de tempo fixo (menor porção de tempo

usada no nó CAN) derivado do período do oscilador do sistema. Nas figuras 2.27 e 2.28 estão ilustrados, respectivamente, o diagrama com os segmentos que formam um *bit time* e o formato de um *time quanta*.

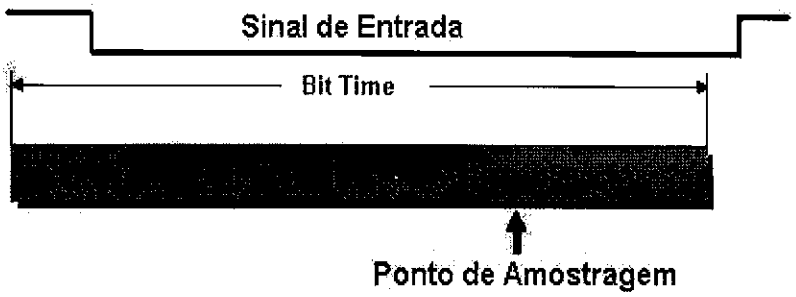

Figura 2.27: Diagrama com os Segmentos que Formam um *Bit Time*.

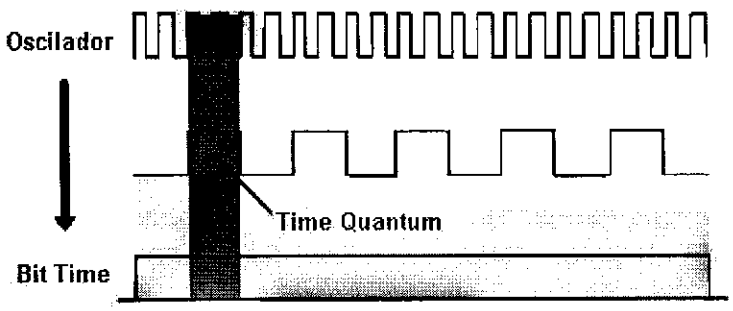

Figura 2.28: Esquema com o Formato de um *Time Quanta*.

A taxa de *bit* nominal é definida na especificação CAN como o número de *bits* por segundo transmitido por um transmissor ideal sem resincronização e pode ser descrita pela equação a seguir:

$$TBN = fbit = 1/Tbit$$

Onde:

TBN é a Taxa de *Bit* Nominal.

Tbit é o *bit time*.

O *bit time* deve ter no mínimo uma duração de 8 TQ e no máximo de 25 TQ. Ele é dado pela seguinte equação:

$$Tbit = T_{Seg\_Sinc} + T_{Seg\_Prop} + T_{Seg\_Fase1} + T_{Seg\_Fase2}$$

Onde:

- T_Seg_Sinc é o tempo de duração em TQ do segmento de sincronização.
- T_Seg_Prop é o tempo de duração em TQ do segmento de propagação.
- T_Seg_Fase1 é o tempo de duração em TQ do segmento de fase 1.
- T_Seg_Fase2 é o tempo de duração em TQ do segmento de fase 2.

As funções de gestão do barramento executadas durante o *bit time*, tais como o ajuste de sincronização de um nó, atraso de compensação da transmissão da rede e a posição dos pontos de amostragem, são definidas pela lógica programável do *bit time* do controlador CAN.

Associado ao *bit time*, temos o ponto de amostragem, tempo de processamento da informação (IPT) e a largura de pulso de sincronização (SJW). A seguir, temos as características de cada segmento do *bit time* e dos parâmetros que estão relacionados a ele:

- **Seg_Sync** - É usado para sincronizar os vários nós do barramento. Espera-se que a borda de descida em uma recepção ocorra neste segmento (caso ideal). Este segmento é fixado em 1 TQ.
- **Seg_Prop** - É um campo de largura variável usado para compensar os tempos de atraso físico entre os nós da rede. Estes atrasos consistem nos tempos de propagação do sinal na linha do barramento e nos tempos de atraso interno aos nós. Este segmento deve ter um tempo de duração entre 1 e 8 TQ.
- **Seg_Fase1** - É usado para compensar o erro de fase na transição e pode ter o seu comprimento aumentado durante a resincronização. Este segmento deve ter um tempo de duração entre 1 e 8 TQ.
- **Seg_Fase2** - É usado para compensar o erro de fase na transição e pode ter o seu comprimento diminuído durante a resincronização. Este segmento deve ter um tempo de duração entre 2 e 8 TQ.

- **Ponto de Amostragem** - É o ponto do *bit time* em que o nível lógico é lido e interpretado. Ele está localizado no fim do segmento de fase 1. Uma exceção para essa regra ocorre se o modo de amostra está configurado para amostrar o *bit* três vezes. Neste caso, o *bit* é amostrado até o fim do segmento de fase 1.

- **Tempo de Processamento da Informação (IPT)** - É o tempo requisitado para determinar o nível de um *bit*. O IPT começa no ponto de amostragem. A sua duração depende do controlador. Para o controlador CAN da Microchip, o tempo de duração é de 2 TQ.

- **Largura do Pulso de Sincronização (SJW)** - O SJW ajusta o tempo necessário para manter uma sincronização com a mensagem transmitida. O tempo de duração varia entre 1 e 4 TQ.

2.15 Sincronismo

Duas técnicas de sincronização são utilizadas para manter um sincronismo entre os nós da rede, são elas:

- **Sincronização Forçada** - É acionada na borda de descida quando o barramento está livre, que é interpretado como o SOF. Essa sincronização inicia a lógica interna do *bit time*.

- **Re-sincronização** - É usado para ajustar o *bit time* enquanto um nó CAN está recebendo uma mensagem, de forma a permitir que a localização do ponto de amostragem seja correta. Quando o transmissor é lento em relação ao receptor, o *bit time* é aumentado. Quando o transmissor é rápido, o bit time é diminuído.

A cada borda de descida inicia-se a lógica do *bit time*. Idealmente, essa borda é esperada no segmento de sincronização, conforme figura 2.27. Porém, pode acontecer devido à alteração de fase, que a borda de descida venha acontecer durante o segmento de fase 1 ou no segmento de fase 2.

Se a borda de descida do sinal de entrada for detectada quando o *bit time* estiver no segmento de fase 1, ver figura 2.29, interpreta-se que o transmissor é lento em relação ao receptor. Neste caso, o segmento de fase 2 é adicionado de SJW pulsos para que o sinal de amostra esteja no ponto adequado (ponto 2). Se não houvesse um sistema de sincronização, o sinal seria amostrado no ponto 1 da figura.

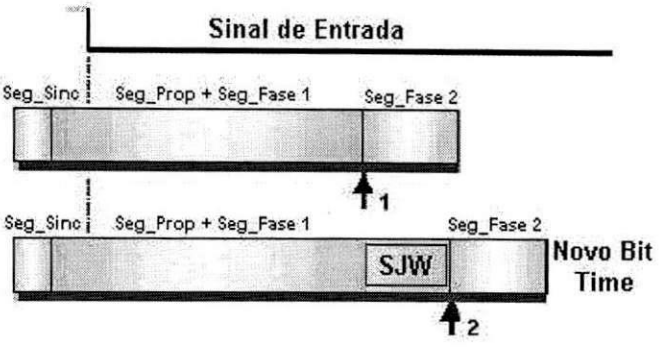

Figura 2.29: Esquema com o Processo de Sincronização com Transmissor Lento.

Se a borda de descida coincidisse com o segmento de fase 2, como ilustrado na figura 2.30, poderíamos interpretar que o transmissor é mais rápido que o receptor. Neste caso, o segmento de fase 2 do *bit* anterior é diminuído de SJW pulsos de forma que o novo *bit time* comece mais cedo do que começaria (ponto 3). Com isso, consegue-se que o ponto de amostragem esteja localizado no ponto 2 ao invés do ponto 1.

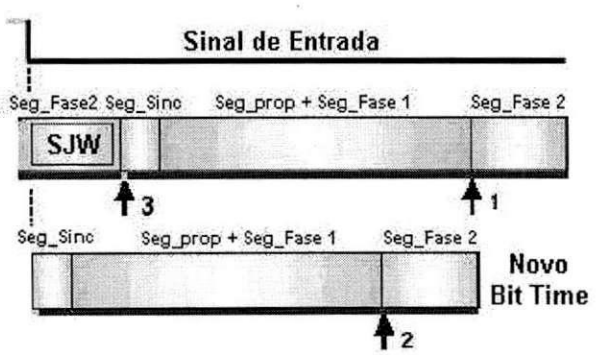

Figura 2.30: Esquema com o Processo de Sincronização com Transmissor Rápido.

2.16 Meio de Transmissão

É possível utilizar diversos meios físicos de transmissão, tais como: par de fios trançados, fibra óptica, rádio freqüência e etc. A maioria das aplicações utiliza um barramento diferencial a dois fios, o CAN_L e o CAN_H. Na figura 2.31 é ilustrada a fiação do barramento.

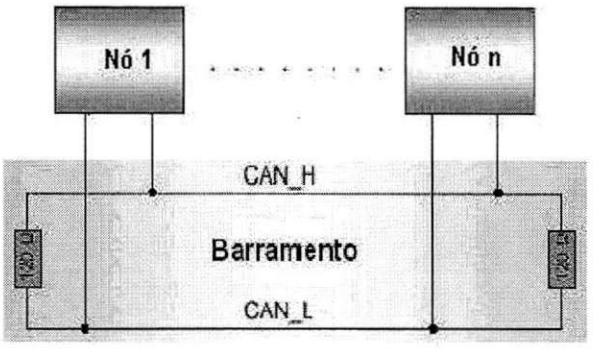

Figura 2.31: Esquema do Barramento Diferencial.

O CAN usa a técnica de sinalização por tensão diferencial, isto é, o nível do sinal, recessivo (quando CAN_H tem uma tensão abaixo de 0.5 v em relação à tensão CAN_L) ou dominante (quando CAN_H for pelo menos 0.9v maior que CAN_L), é determinado pela diferença de tensão entre CAN_L e CAN_H. Na figura 2.32 é ilustrada essa técnica de sinalização. Essa técnica de fiação consegue minimizar efeitos de interferências eletromagnéticas, pois essa interferência vai afetar ambos os fios da mesma forma, permanecendo a tensão diferencial igual. Na figura 2.33 é ilustrado o processo de imunidade as interferências magnéticas.

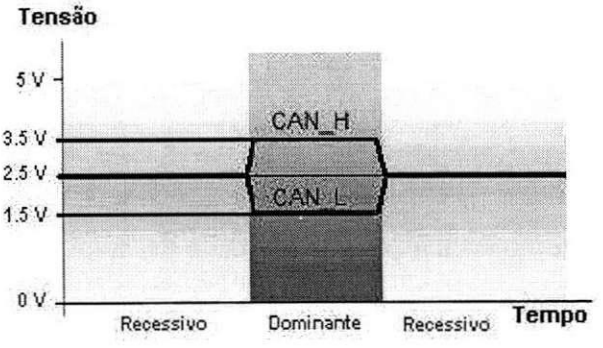

Figura 2.32: Técnica de Sinalização Diferencial.

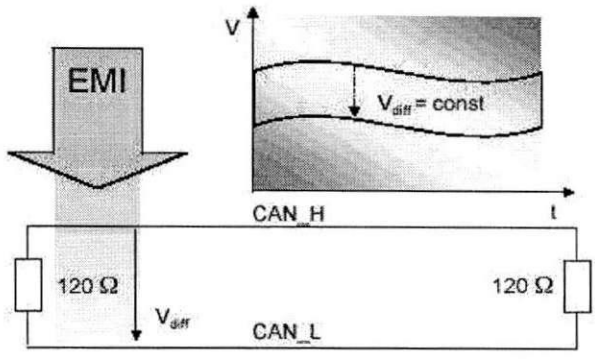

Figura 2.33: Processo de Imunidade as Interferências Eletromagnéticas.

2.17 *Transceiver*

Para se fazer à *interface* entre o controlador CAN e o barramento, é necessário que exista um *transceiver* para fazer a codificação do sinal do controlador para o sinal de tensão diferencial, do barramento. Na figura 2.34 é ilustrado esse processo.

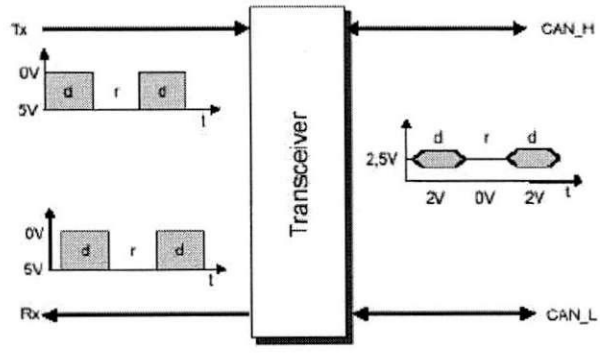

Figura 2.34: Esquema de Funcionamento do *Transceiver*.

2.18 Tolerâncias de Falhas no Barramento

A rede CAN pode funcionar em condições extremamente adversas, e os mecanismos de detecção de erros extremamente completos garantem que os erros que possam ocorrer durante a transmissão sejam detectados. Assim a rede continua operacional mesmo nas seguintes situações:

- Um dos condutores partidos, uma vez que pode utilizar somente um dos condutores para efetuar a comunicação, deixando no caso de fazer a verificação *bit* a *bit*;
- Um dos condutores em curto com a fase. Então podemos utilizar somente um dos condutores, análogo à situação anterior;
- Um dos condutores em curto com a terra. Então podemos utilizar somente um dos condutores, análogo a ambas as situações anteriores;
- Os dois condutores em curto, entre si. Neste caso a rede funciona parcialmente, uma vez que fica dividida (pelo curto circuito) em duas sub-redes podendo somente as estações de uma sub-rede comunicar com as estações dessa sub-rede.

2.19 Arquitetura dos Controladores

Apesar da especificação CAN ser bem direcionada, podemos ter variações de certos itens do controlador. A seguir veremos essas variações do controlador.

2.19.1 Integração

Podemos ter três variações quanto à integração do controlador, são elas:

- **Stand-Alone** - O controlador CAN é separado do microcontrolador e do *transceiver*. É desenvolvido para fazer *interface* com diversos processadores diferentes permitindo reutilização de código. Na figura 2.35 é ilustrado o diagrama de blocos de um módulo CAN com uma integração do tipo *stand-alone*.

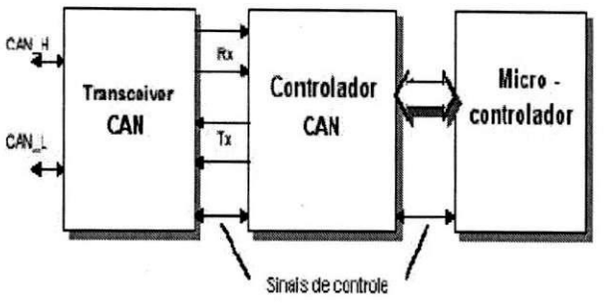

Figura 2.35: Diagrama de Blocos de um Módulo CAN *Stand-Alone*.

- **Integrado** - O controlador CAN é integrado com o microcontrolador. Nesse caso, teremos uma redução na carga do microcontrolador. Um fator crítico é a quantidade de tempo requisitado para ler e escrever nos periféricos CAN. Neste caso, os registradores CAN são endereçados usando um barramento de dados e de endereço interno de alta velocidade. No caso de um módulo *stand - alone*, esse barramento de dados e de endereço é externo, ou utiliza-se uma comunicação serial, que é possui uma velocidade bem menor. Na figura 2.36 é ilustrado um diagrama de blocos de um módulo CAN com uma integração do tipo integrado.

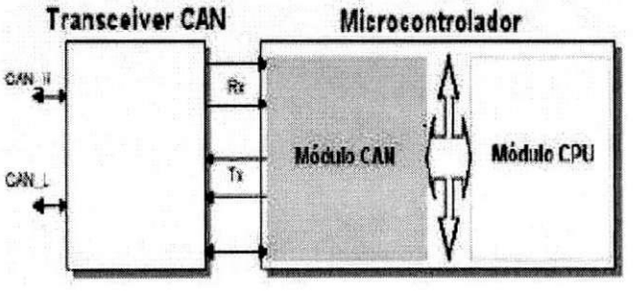

Figura 2.36: Diagrama de Blocos de um Módulo CAN Integrado.

- ***Single-Chip*** - Integra em um único *chip* o *transceiver*, o controlador CAN e o microcontrolador. O principal problema desse tipo de integração é a combinação de diferentes tecnologias e um único *chip*. Na figura 2.37 é ilustrado o diagrama de blocos de um módulo CAN do tipo *single-chip*.

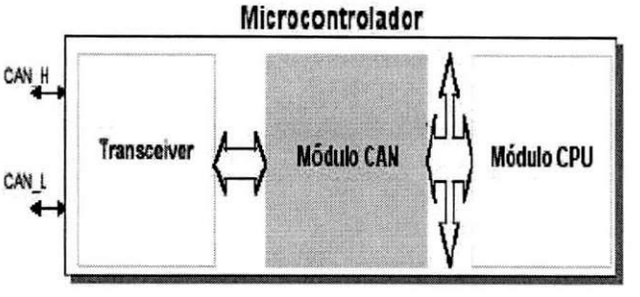

Figura 2.37: Diagrama de Blocos de um Módulo CAN *Single-Chip*.

2.19.2 Armazenamento de Mensagens

Quanto ao armazenamento de mensagens existem duas diferentes versões:

- **CAN básico** - Todas as mensagens que existam no barramento, são verificadas individualmente pelo microcontrolador. Isto resulta na ocupação da CPU para verificar todas as mensagens, em vez de processar apenas as relevantes, o que tende a limitar a taxa de transmissão a 250 Kbps;
- **CAN completo** - Existe um filtro de aceitação que permite ao controlador recusar mensagens irrelevantes, através da verificação dos identificadores, libertando o CPU para processar apenas mensagens consideradas relevantes o que permite conseguir taxas de transmissão mais elevadas (1 Mbps).

2.19.3 Filtro de Aceitação

A função do filtro de aceitação é verificar se o dado que está sendo recebido é relevante ou não para nó. O filtro é implementado comparando-se o identificador da mensagem que está chegando, com os filtros existentes. O uso de filtros no controlador libera o microcontrolador da tarefa de seleção de dados. Existem duas implementações de filtro, são elas:

- **Filtro Simples** - Uma máscara é usada para comparar os *bits* de identificação que estão no barramento. Por ter somente uma máscara, o nó receberá somente uma mensagem. Na figura 2.38 é ilustrado esse tipo de filtro.

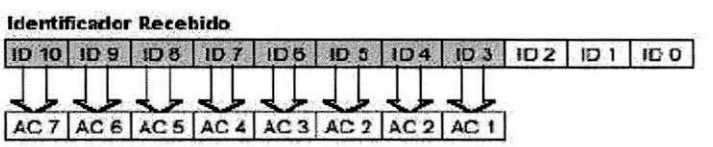

Figura 2.38: Esquema de um Filtro Simples.

- **Filtro Múltiplo** - Várias máscaras são usadas na comparação da mensagem de entrada. Desta forma o nó poderá receber várias mensagens. Na figura 2.39 é ilustrado esse filtro.

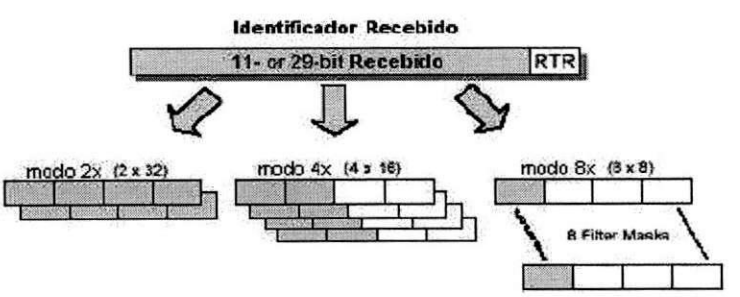

Figura 2.39: Esquema de um Filtro Múltiplo.

Capítulo 3

Arquitetura e Descrição do Sistema Implementado

3.1 Arquitetura do Sistema

O sistema proposto é constituído por três nós CAN (Nó CAN_1, Nó CAN_2 e Nó CAN_3), os quais se comunicam por um barramento constituído por dois fios de 1,5 metros de comprimento a uma taxa de transmissão de aproximadamente 690 Kbps.

Os nós CAN 1 e 2 possuem cada, um sensor de temperatura LM35 (ST1 e ST2), um potenciômetro de 10 KΩ (P1 e P2), circuitos de condicionamento, e apenas o nó CAN_1, um *cooler* que é ligado/desligado de acordo com o valor da temperatura medido pelo sensor ST1. O nó CAN_3 é apenas um nó receptor de informações. Ele gera dois sinais PWM com ciclos de trabalho variáveis controlados pelos potenciômetros P1 e P2.

As informações adquiridas pelos nós 1 e 2 através do barramento CAN e pelos seus próprios dispositivos são transmitidos utilizando a *interface* de comunicação serial para os seus respectivos computadores (PC1 e PC2) e visualizados através do aplicativo *hyper terminal*. Na figura 3.1 apresenta-se um diagrama de blocos do sistema implementado.

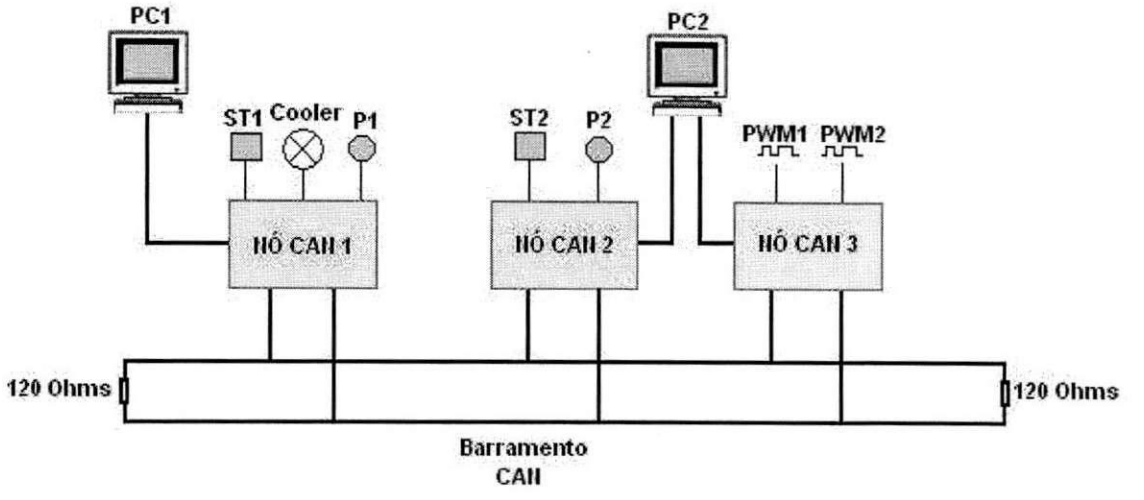

Figura 3.1: Diagrama de Blocos do Sistema Implementado.

3.2 Descrição do Funcionamento do Sistema

Os sensores de temperatura ST1 e ST2 convertem o valor de temperatura ambiente em sinal de tensão, e em seguida este sinal é condicionado e posteriormente transmitido para a entrada do conversor A/D do microconversor ADuC812. Após o processo de conversão do sinal de tensão, alguns calcúlos são realizados para transformarem os valores de tensão em valores de temperatura. Esses valores de temperatura são transmitidos pela *interface* de comunicação serial para o PC do respectivo nó CAN (Nó CAN 1 e 2) para serem visualizados no aplicativo *hyper terminal*. Os valores de temperatura medidos pelos sensores ST1 e ST2 serão transmitidos do nó CAN_1 para o nó CAN_2, e vice-versa, pelo barramento CAN, quando a respectiva chave de requisição de transmissão da informação de temperatura estiver fechada. Quando o valor de temperatura medido pelo sensor ST1 (Nó CAN_1) for maior que 30°C, o nó CAN_2 enviará uma mensagem CAN indicando que o *cooler* deverá ser acionado. Quando a temperatura for inferior a 30°C, o nó CAN_2 enviará uma mensagem CAN indicando que o *cooler* deverá ser desligado. Esses valores de temperatura enviados de um nó para o outro, e o *status* do *cooler* também são visualizados dos através do aplicativo *hyper terminal*.

Os potenciômetros P1 e P2 são utilizados para ajustar o ciclo de trabalho dos sinais PWM 1 e 2, respectivamente, gerados no nó CAN_3. P1 e P2 estão conectados nas entradas dos conversores A/D dos seus microconversores, os quais convertem níveis de tensão entre 0 e 5 Volts regulados pelos potenciômetros. Após o processo de conversão do sinal em tensão, esses valores digitais são transmitidos pela *interface* de comunicação serial para o PC do respectivo nó CAN (Nó CAN 1 e 2) para serem visualizados no aplicativo *hyper terminal*. Os valores dos potenciômetros serão transmitidos para o nó CAN_3 pelo barramento CAN, quando a respectiva chave de requisição de transmissão da informação do potenciômetro estiver fechada. Então, o nó CAN_3 ao receber essas informações, ajustam a largura do ciclo de trabalho dos sinais PWM. Na figura 3.2 apresenta-se um diagrama com as possíveis transmissões e recepções de mensagens CAN do Sistema Implementado.

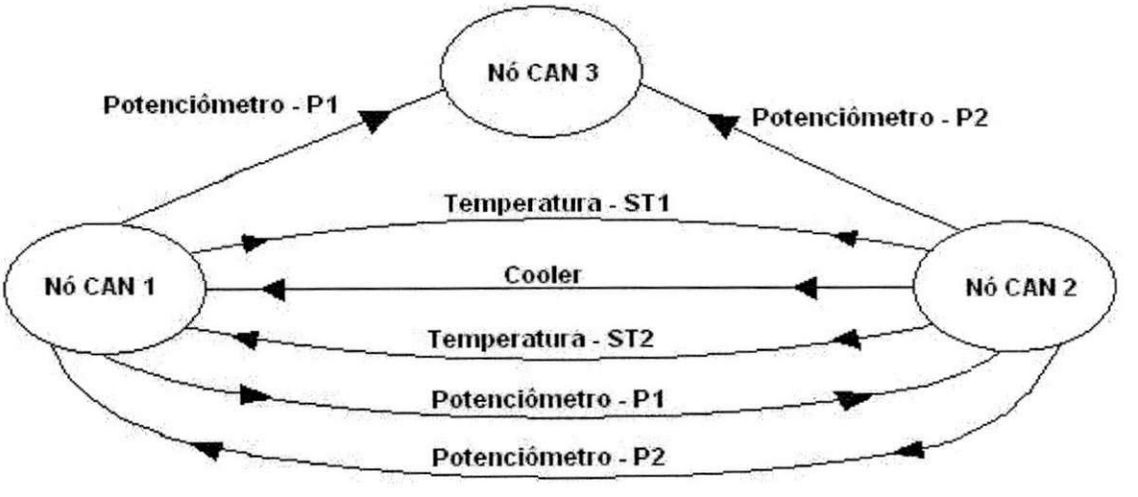

Figura 3.2: Diagrama com as possíveis transmissões e recepções de mensagens CAN do Sistema Implementado.

3.3 Programas dos Nós CAN

Os programas dos nós CAN 1 e 2 foram editados em linguagem de programação C, utilizando o *software Keil uVision* [12] e foram transferidos para os microconversores via a *interface* de comunicação serial utilizando o aplicativo WSD (*Windows Serial Downloader*). O programa do nó CAN 3 foi também escrito em linguagem C [13]. Porém, utilizamos o *software Code Composer* [14] [15] para editar, gerar e transferir o programa para o DSP TMS320LF2407A [16] [17] [18]. Nas figuras 3.3, 3.4 e 3.5 estão ilustrados, respectivamente, o ambiente de programação *Keil uVision*, o aplicativo WSD e o ambiente de programação *Code Composer*.

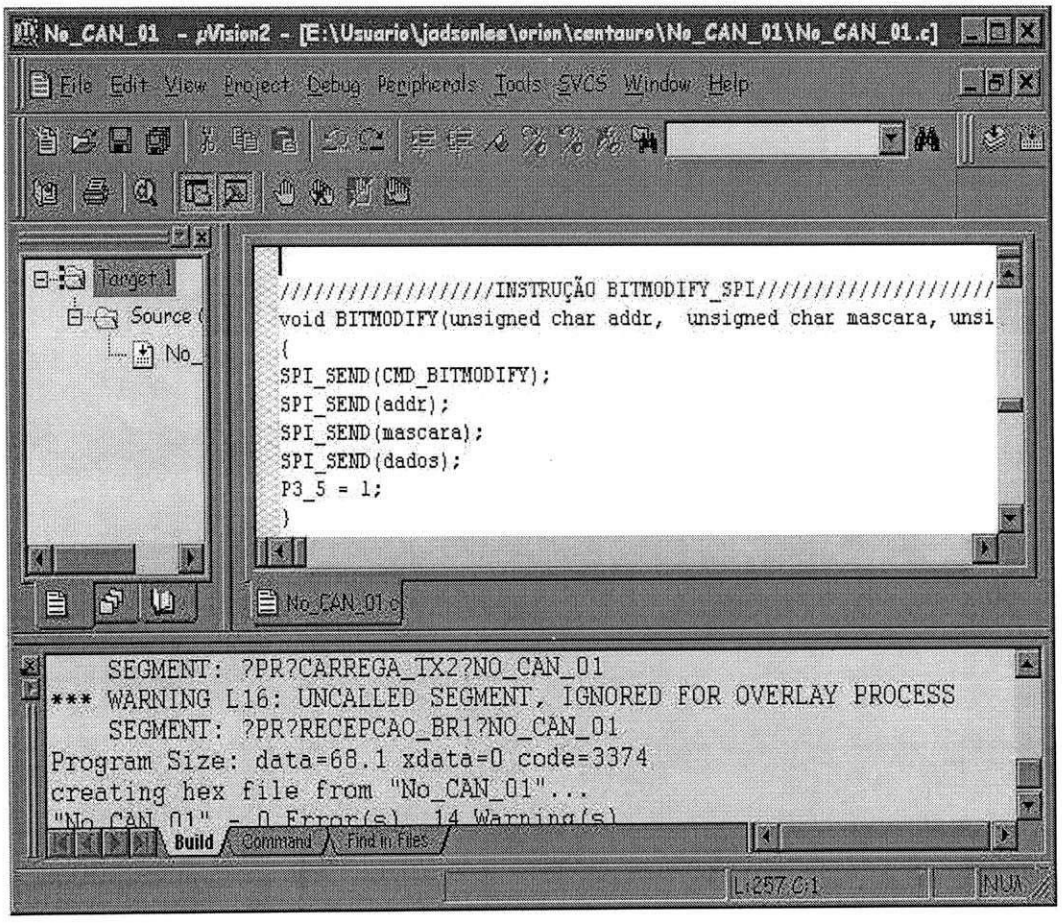

Figura 3.3: Tela do *Software Keil uVision*.

Figura 3.4: Tela do Aplicativo WSD.

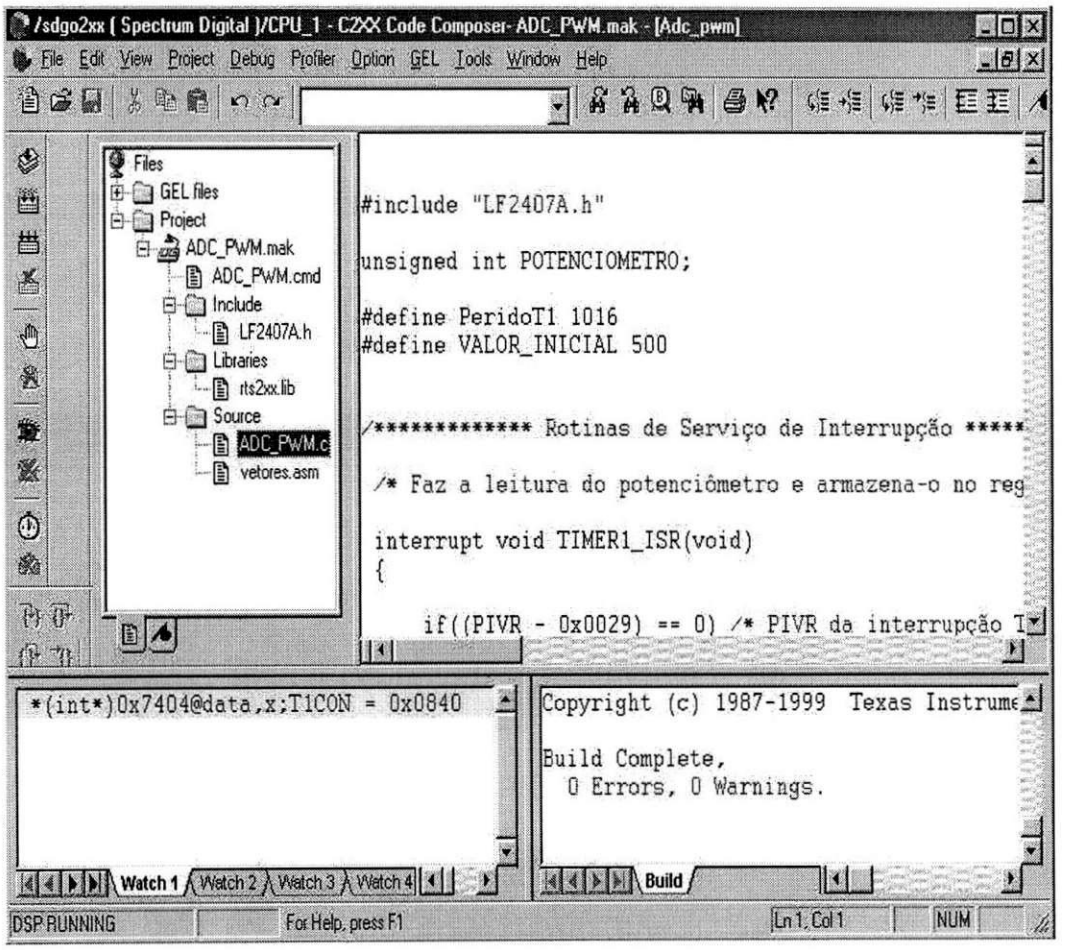

Figura 3.5: Tela do *Software Code Composer*.

3.4 Arquitetura dos Nós CAN

Os nós CAN 1 e 2 possuem uma arquitetura do tipo *Stand-Alone*. Cada um deles é constituído por um *transceiver* CAN (MCP2551) [20], um controlador CAN (MCP2510) [21] [22] e um microconversor (ADuC812) [23][24]. O diagrama elétrico desses dois nós CAN são idênticos. Na figura 3.6 apresenta-se o diagrama elétrico simplificado do nó CAN_1.

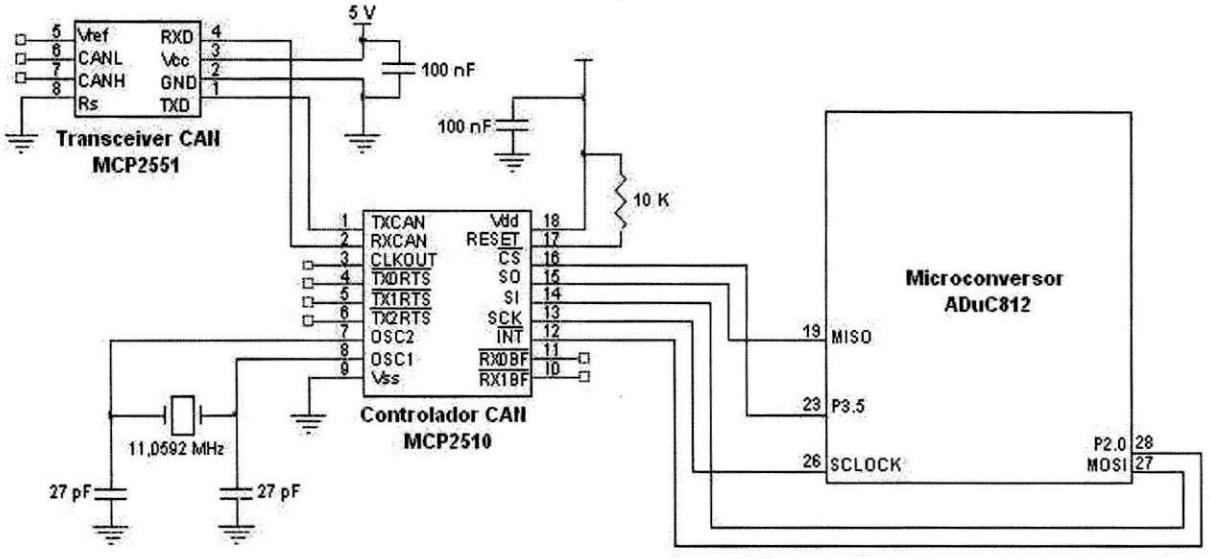

Figura 3.6: Diagrama Elétrico Simplificado do Nó CAN_1.

O nó CAN_3 possui uma arquitetura do tipo integrada. Ele é constituído por um *transceiver* CAN (MCP2551), circuitos de condicionamento e pelo Processador Digitais de Sinais TMS320LF2407A [16] [17] [18] [19]. Na figura 3.7 apresenta-se o diagrama elétrico simplificado do nó CAN_3.

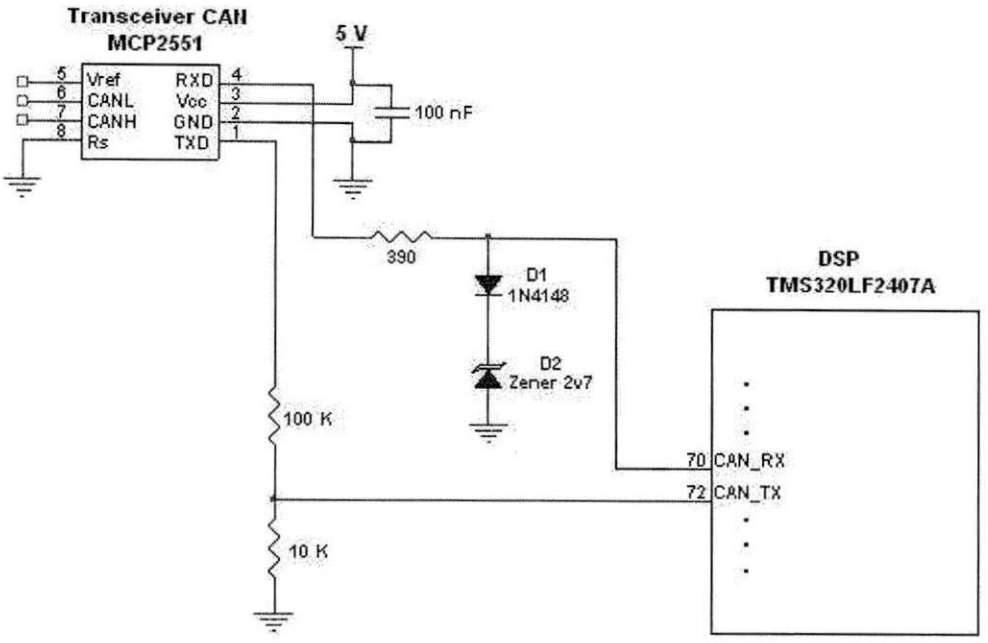

Figura 3.7: Diagrama Elétrico Simplificado do Nó CAN_3.

3.5 Comunicação entre o Controlador MCP2510 e o ADuC812

A comunicação entre o controlador CAN e o ADuC812 é feita pela *interface* SPI [25]. A SPI é uma *interface* serial síncrona na qual transfere e recebe dados (8 *bits*) simultaneamente, isto é, *full duplex*, de um dispositivo mestre para vários dispositivos escravos [24]. A sua *interface* consiste de 4 fios: CLK (*clock*), que determina a velocidade e o sincronismo da transferência e recepção de dados. No dispositivo mestre, o pino CLK é sempre uma saída, e no dispositivo escravo é sempre uma entrada; MOSI (*Master Out Slave In*), é o pino no qual os dados são transferidos do mestre para o escravo; MISO (*Master In Slave Out*), é o pino onde os dados são transferidos do escravo para o mestre; e o CS (*Chip Select*), é o pino que habilita a transferência de dados quando o seu nível for baixo. Esse pino no modo mestre é uma saída, e no modo escravo é uma entrada. Neste projeto, o controlador CAN está configurado como escravo e o microconversor como mestre. O MCP2510 possui 6 funções SPI, são elas: leitura, escrita, leitura de *status*, modificação de *bit*, *reset* e requisição de envio. A seguir veremos ilustrações dos sinais nos pinos (SCLK, CS, MOSI e MISO) da SPI para cada uma das instruções citadas.

3.5.1 Instrução de Leitura

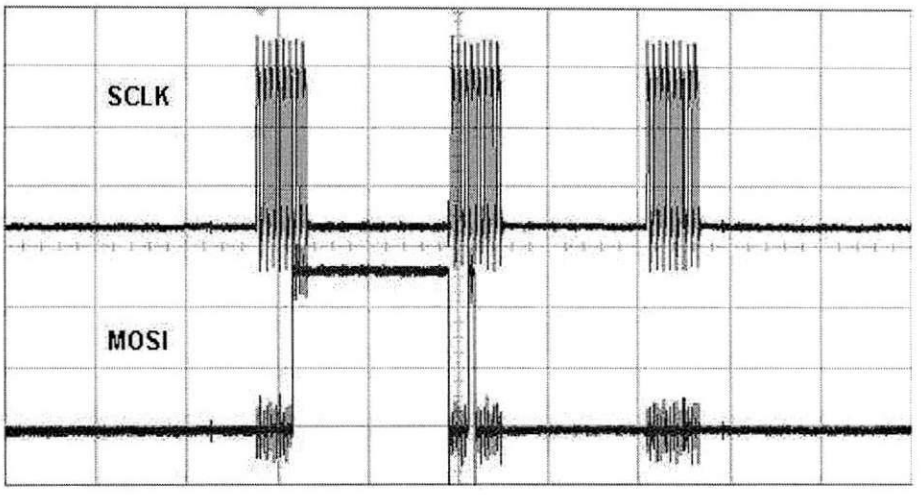

Figura 3.8: Tela com os Sinais dos Pinos SCLK e MOSI em uma Instrução de Leitura.

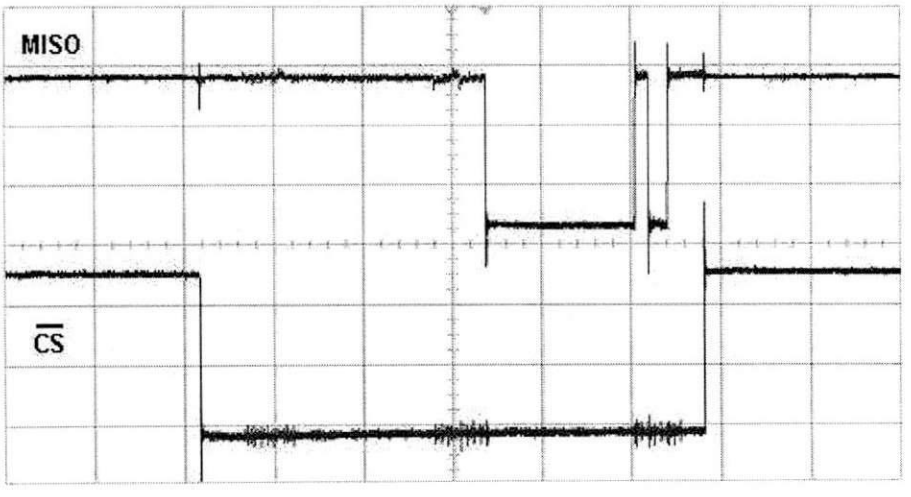

Figura 3.9: Tela com os Sinais dos Pinos MISO e CS em uma Instrução de Leitura.

3.5.2 Instrução de Escrita

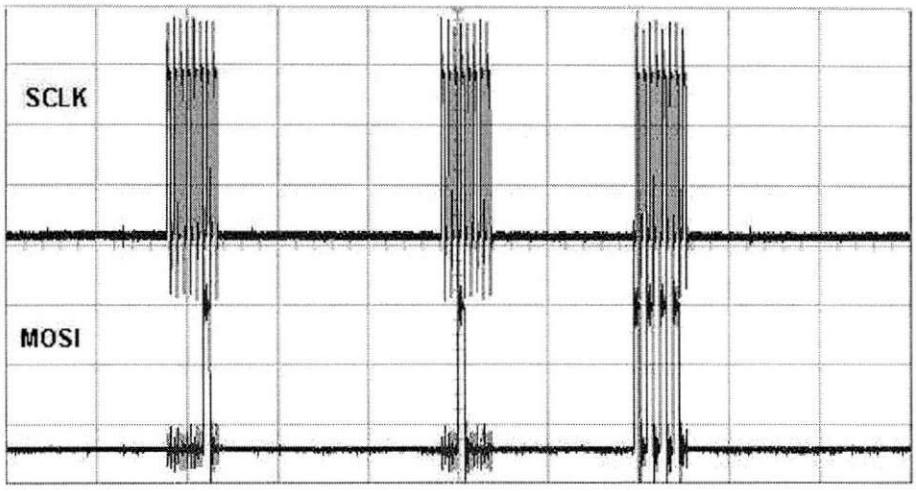

Figura 3.10: Tela com os Sinais dos Pinos SCLK e MOSI em uma Instrução de Escrita.

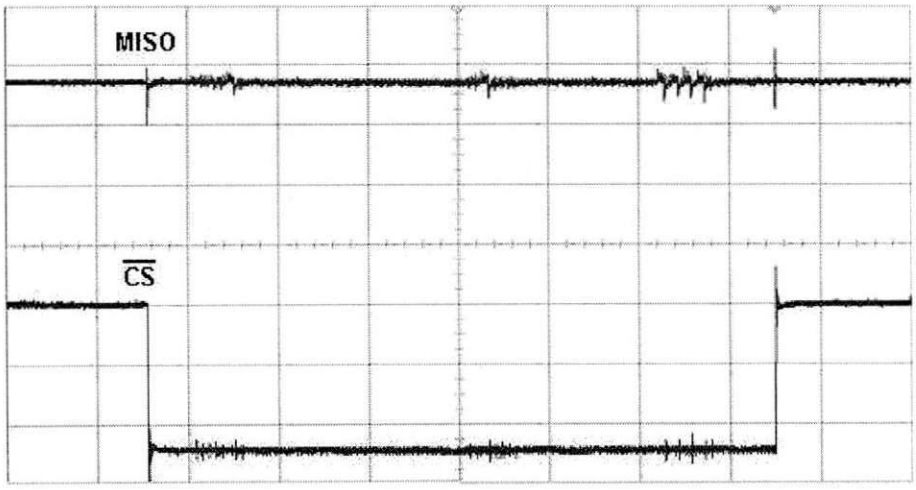

Figura 3.11: Tela com os Sinais dos Pinos MISO e CS em uma Instrução de Escrita.

3.5.3 Instrução de Leitura de *Status*

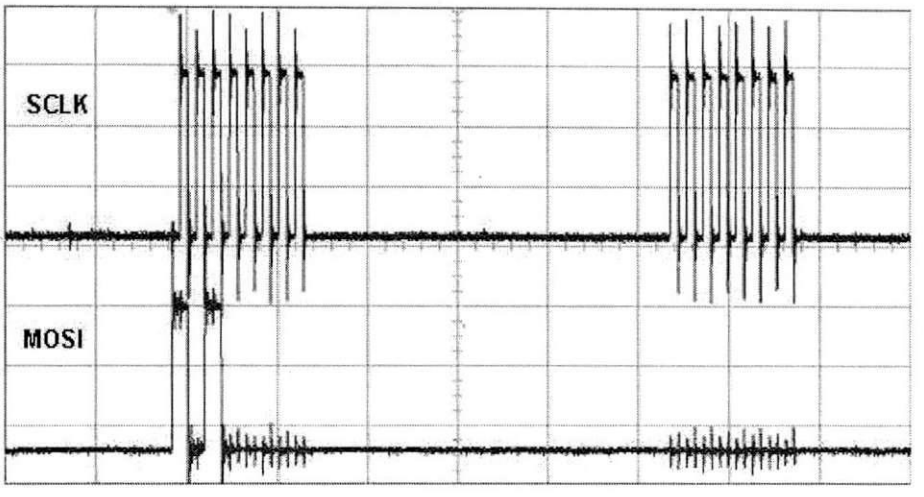

Figura 3.12: Tela com os Sinais dos Pinos SCLK e MOSI em uma Instrução de Leitura de *Status*.

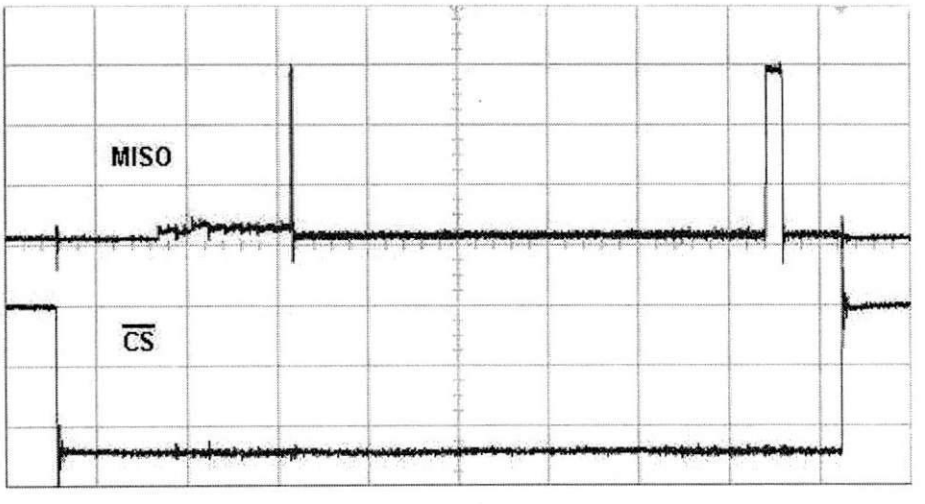

Figura 3.13: Tela com os Sinais dos Pinos MISO e CS em uma Instrução de Leitura de *Status*.

3.5.4 Instrução de Modificação de *Bit*

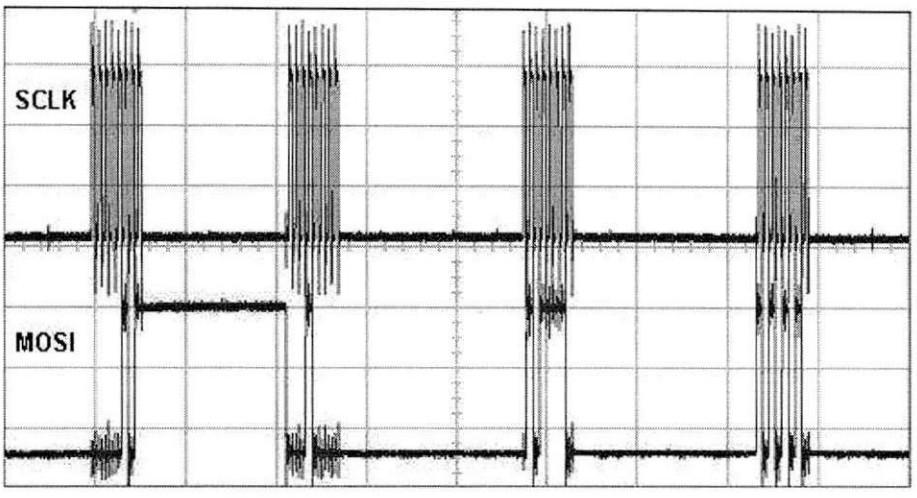

Figura 3.14: Tela com os Sinais dos Pinos SCLK e MOSI em uma Instrução de Modificação de *Bit*.

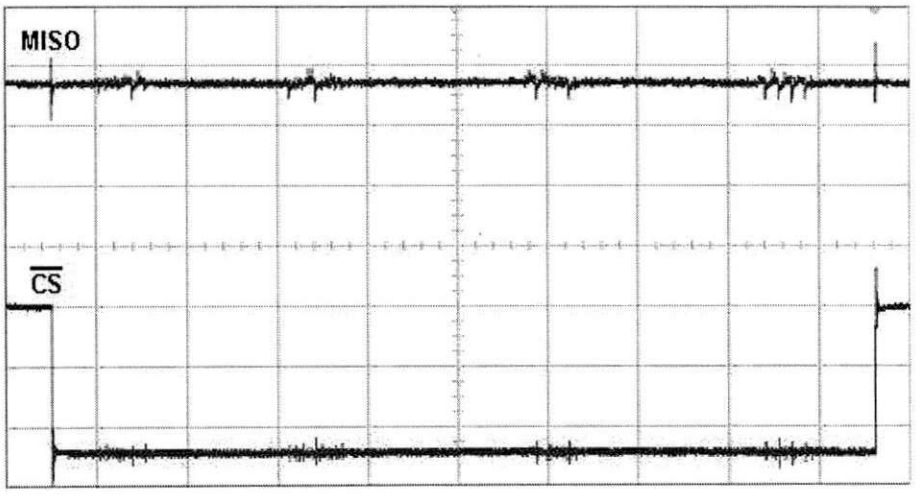

Figura 3.15: Tela com os Sinais dos Pinos MISO e CS em uma Instrução de Modificação de *Bit*.

3.5.5 Instrução de *Reset*

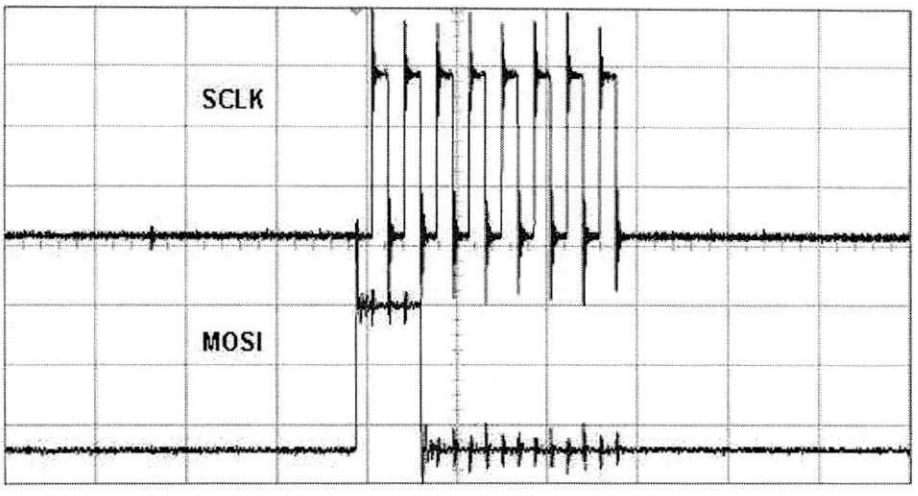

Figura 3.16: Tela com os Sinais dos Pinos SCLK e MOSI em uma Instrução de *Reset*.

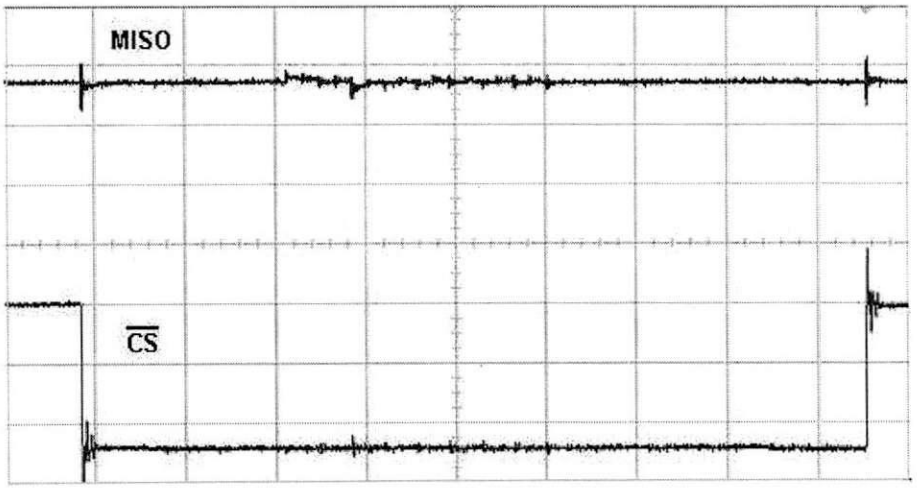

Figura 3.17: Tela com os Sinais dos Pinos MISO e CS em uma Instrução de *Reset*.

3.5.6 Instrução de Requisição de Envio do *Buffer* 0

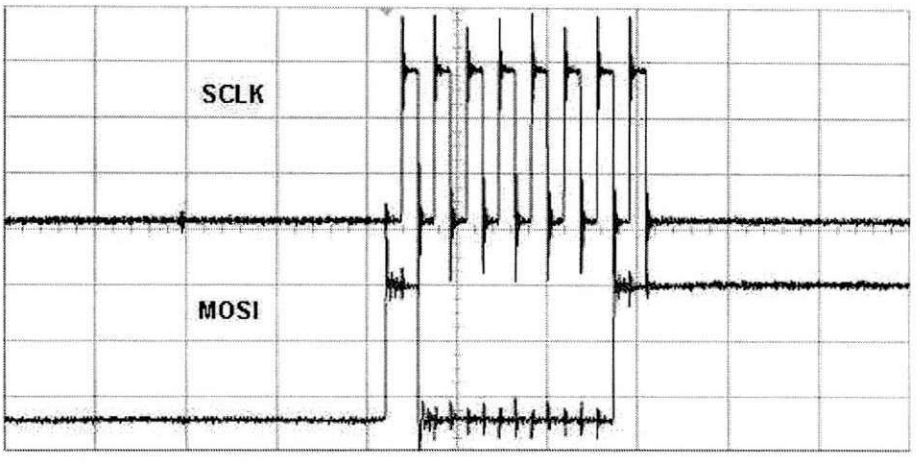

Figura 3.18: Tela com os Sinais dos Pinos SCLK e MOSI em uma Instrução de Requisição de Envio do *Buffer* 0.

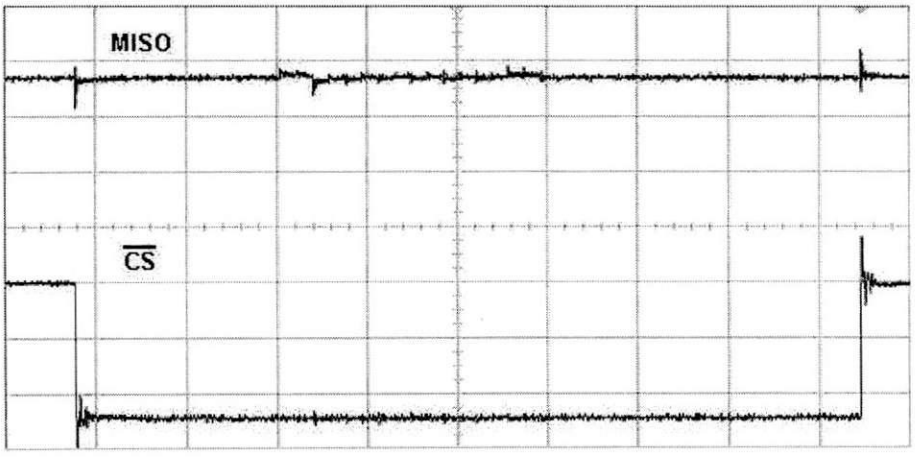

Figura 3.19: Tela com os Sinais dos Pinos MISO e CS em uma Instrução de Requisição de Envio do *Buffer* 0.

3.5.7 Instrução de Requisição de Envio do *Buffer* 1

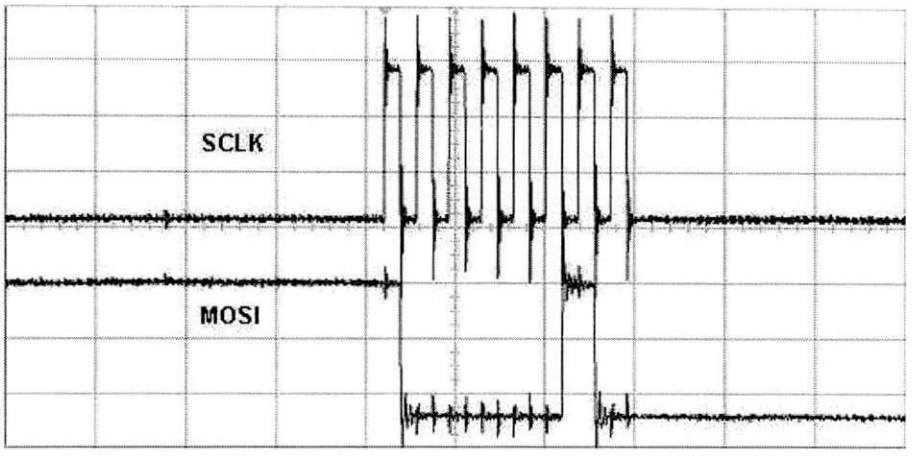

Figura 3.20: Tela com os Sinais dos Pinos SCLK e MOSI em uma Instrução de Requisição de Envio do *Buffer* 1.

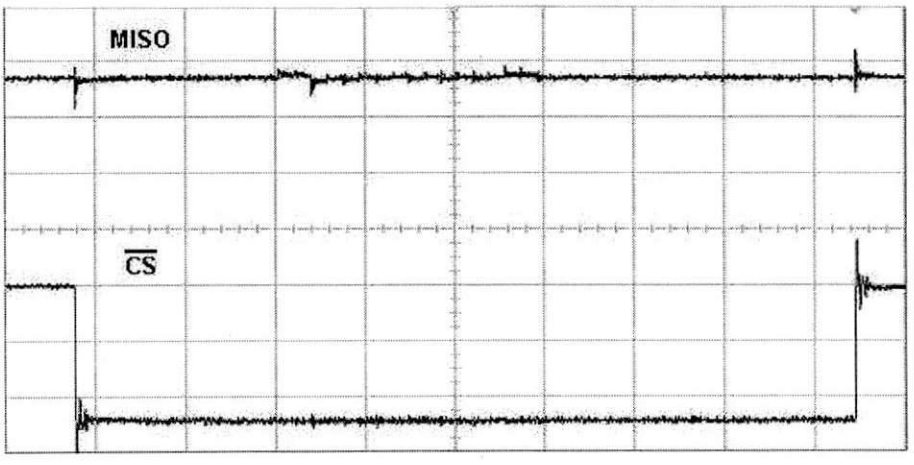

Figura 3.21: Tela com os Sinais dos Pinos MISO e CS em uma Instrução de Requisição de Envio do *Buffer* 0.

3.5.8 Instrução de Requisição de Envio do *Buffer* 2

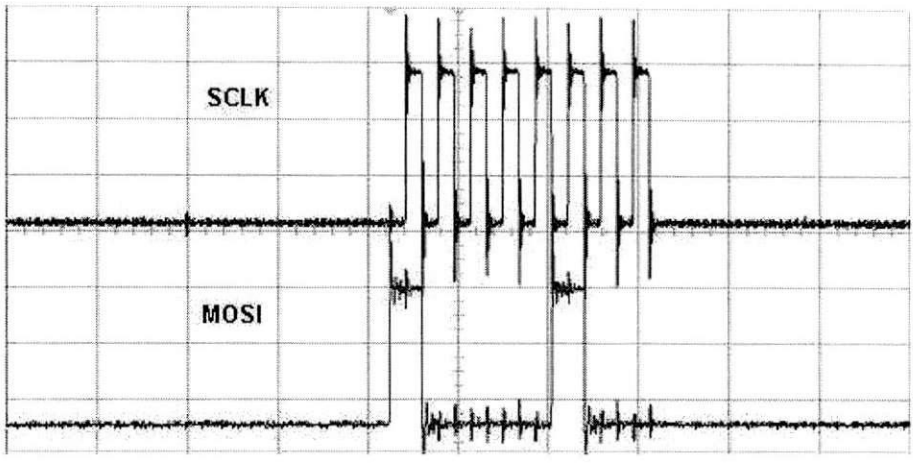

Figura 3.22: Tela com os Sinais dos Pinos SCLK e MOSI em uma Instrução de Requisição de Envio do *Buffer* 2.

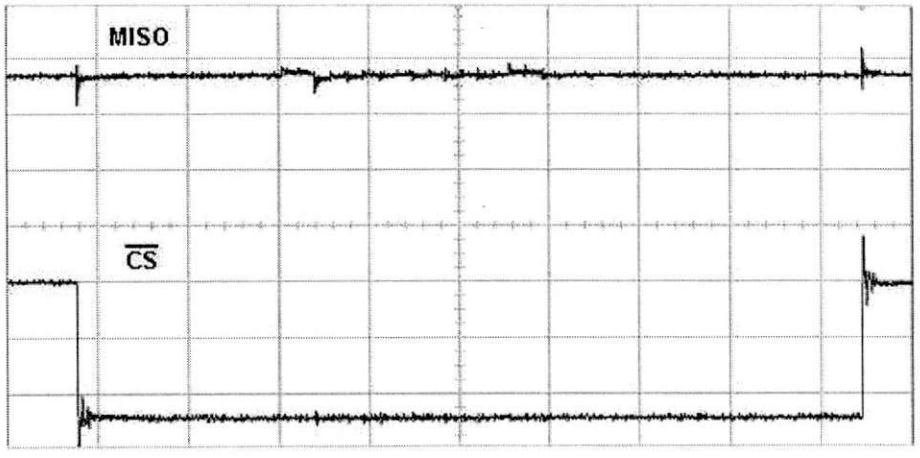

Figura 3.23: Tela com os Sinais dos Pinos MISO e CS em uma Instrução de Requisição de Envio do *Buffer* 2.

3.6 Mensagens CAN

Neste projeto, todas as mensagens possuem formato padrão. Os pacotes de dados contém as informações que serão enviadas de um nó para outro. Na figura 3.24 está ilustrado os sinais de um pacote de dados transmitindo um valor de temperatura do nó CAN1 para o nó CAN2, e o *bit* de reconhecimento da mensagem recebido pelo nó CAN1.

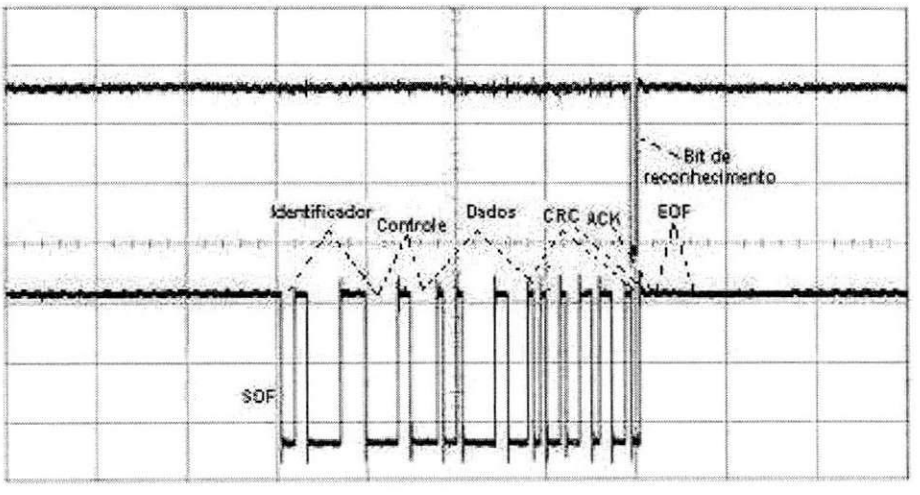

Figura 3.24: Tela com as Informações de um Pacote de Dados e o *Bit* de Reconhecimento.

Capítulo 4

Resultados Obtidos

Os valores de temperatura medidos pelos sensores, o atual *status* do *cooler* e os valores dos potenciômetros foram adquiridos e transmitidos de um nó para o outro pelo barramento CAN e em seguida transferidos pela *interface* de comunicação serial para um PC e visualizados com o auxílio do aplicativo *hyper terminal*. Nas figuras seguintes, apresentam-se telas do *hyper terminal* com as informações das variáveis do sistema.

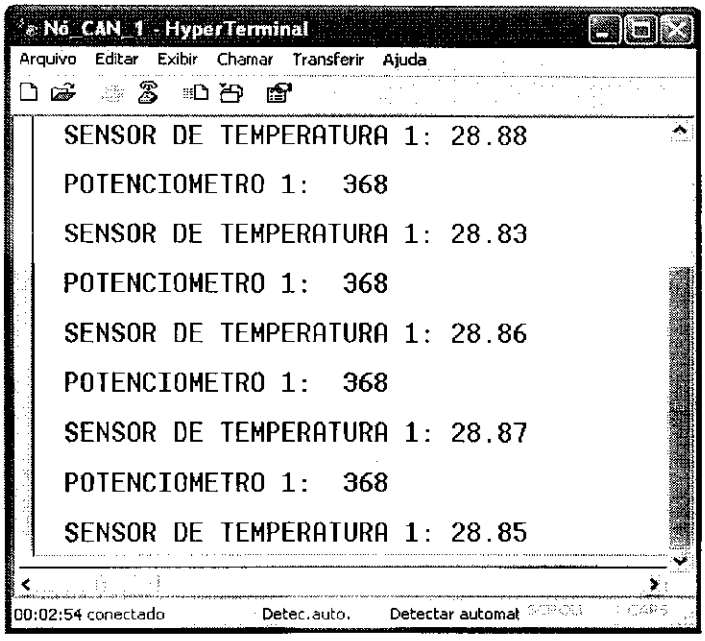

Figura 4.1: Tela do *Hyper Terminal* do PC no Nó CAN1 com as Informações do Sensor ST1 e do Potenciômetro P1.

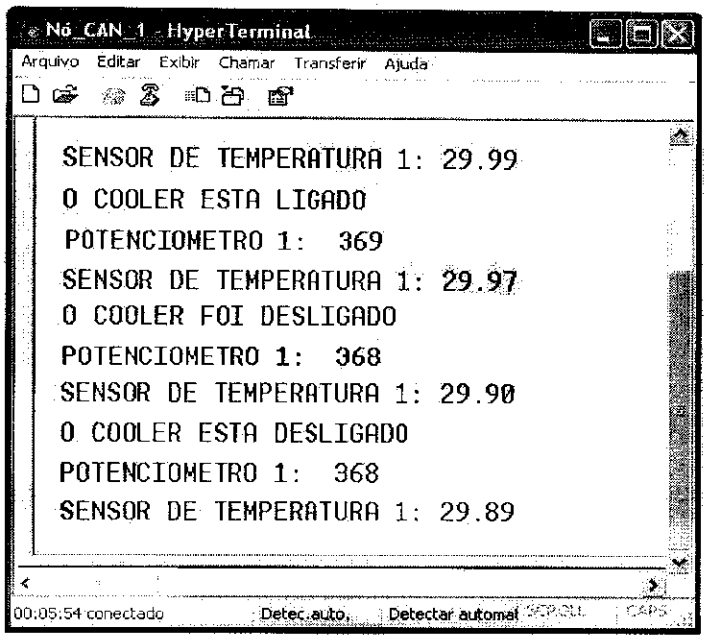

Figura 4.2: Tela do *Hyper Terminal* do PC no Nó CAN1 com as Informações do Sensor ST1, do Potenciômetro P1 e do *Cooler*.

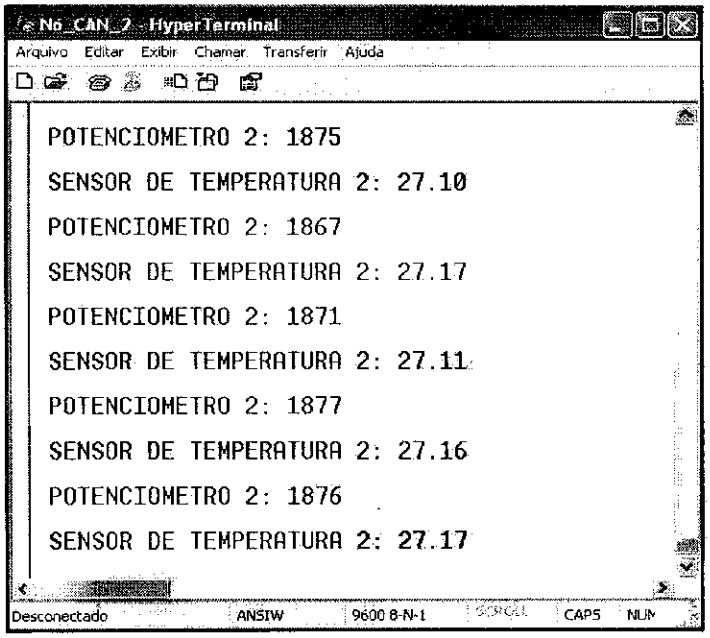

Figura 4.3: Tela do *Hyper Terminal* do PC no Nó CAN2 com as Informações do Sensor ST2 e do Potenciômetro P2.

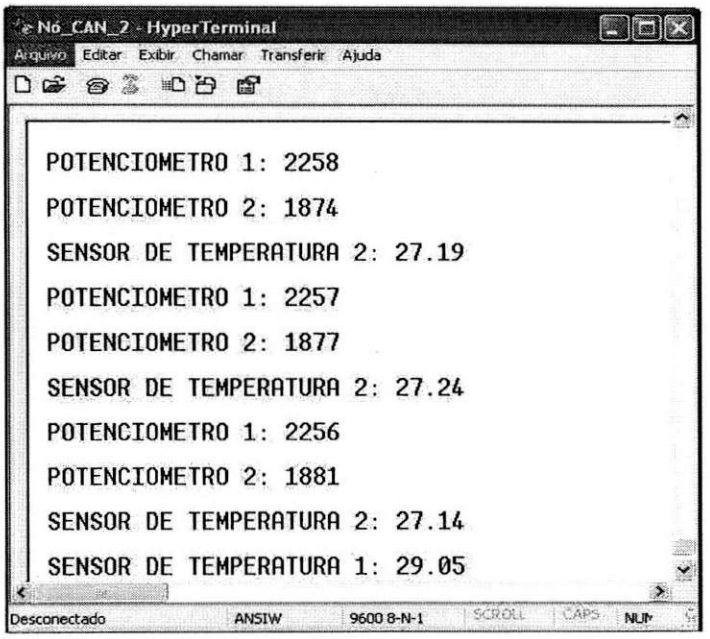

Figura 4.4: Tela do *Hyper Terminal* do PC no Nó CAN2 com as Informações do Sensores ST1 e ST2 e dos Potenciômetros P1 e P2.

Os dois sinais PWM gerados no nó CAN_3 com ciclos de trabalho controlados pelos potenciômetros P1 e P2 foram visualizados com o auxílio de um osciloscópio. Na figura 4.5 apresenta-se a tela do osciloscópio com os sinais PWM1 e PWM2 com diferentes ciclos de trabalho.

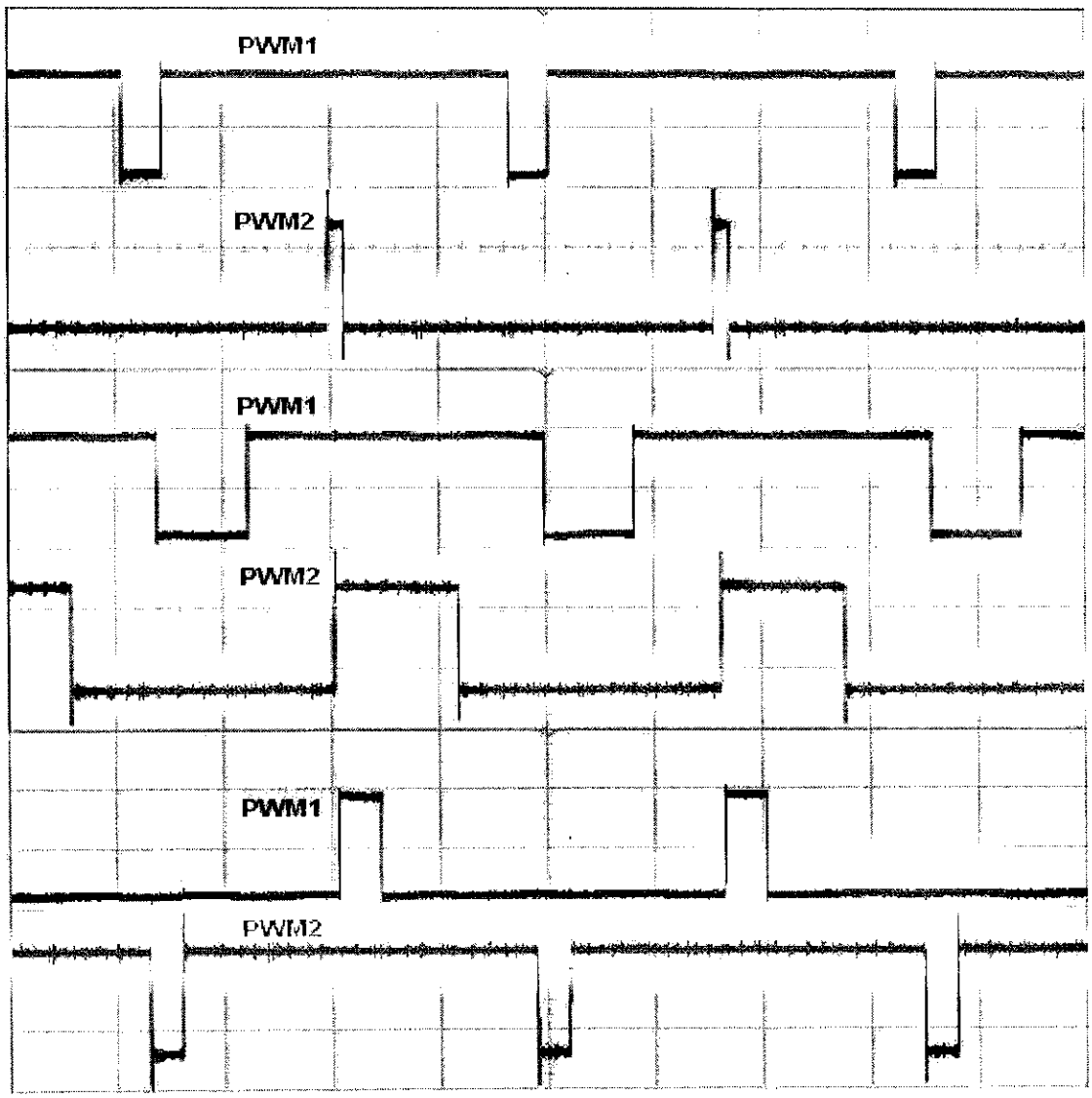

Figura 4.5: Tela do Osciloscópio com os Sinais PWM1 e PWM 2 com diferentes ciclos de trabalho.

Capítulo 5

Conclusões

Com o sistema proposto foi possível observar as vantagens que se obtêm utilizando uma arquitetura distribuída tais como: robustez, confiabilidade, velocidade e possibilidade de expansão para novos pontos de controle.

Devido às características oferecidas pelo protocolo de comunicação CAN, o sistema projetado ofereceu boas condições para que subsistemas inteligentes se comunicassem. A comunicação entre os nós e entre o controlador CAN e o ADuC812 foram bastante eficiente e robusta, proporcionando uma confiabilidade nas informações.

Houve a necessidade de um circuito de condicionamento entre o *transceiver* CAN MCP2551 e o DSP TMS320LF2407A, pois o DSP opera a uma tensão 3,3 Volts e o *transceiver* não funciona bem com essa tensão. O *transceiver* CAN utilizado opera com uma tensão de 5 Volts.

Os valores de temperatura medidos com o sensor LM35 se mostraram satisfatórios, pois foram comparados com outros valores medidos por um termômetro digital e o erro percentual obtido foi de aproximadamente 1.5%.

A resposta de controle do *cooler* e dos ciclos de trabalho dos PWM respeitaram os requisitos exigidos no projeto do sistema.

Agradecimentos

Os autores agradecem o apoio financeiro fornecido pelo CNPq - FINEP (Rede 10/07) para elaboração deste projeto.

Referências Bibliográficas

[1] L. B. Fredriksson, Controller Area Network and the protocol CAN for machine control systems, Mechatronics Vol.4 No.2 pp. 159-192, Sweden, 1994.

[2] Robert Bosch GmbH, CAN Especification, Version 2.0, Setembro de 1991.

[3] K. Pazul, Controller Area Network (CAN) basics, AN713, Microchip Technology, USA, Inc, 1999.

[4] P. Richards, A CAN physical layer discussion, AN228, Microchip Technology, USA, Inc, 2002.

[5] L. C. E. Bona, E. P. Duarte Jr., e K. V. O. Fonseca, Diagnóstico distribuído hierárquico em tempo real, Workshop de tempo real (WTR'2003), Natal, RN, Brazil, 2003.

[6] M. R. Benedetti e J. C. Netto, CANopen aplicado a sistemas distribuídos em tempo real no setor de transmissão de energia elétrica, Workshop de tempo real (WTR'2003), Natal, RN, Brazil, 2003.

[7] J. Kaiser, P. Schaeffer, ICU - A smart optical sensor for direct control, Proc. IEEE International Conference on Mechatronics and Machine Vision in Pratic, Hong Kong, 2001.

[8] D. Mancini, P. Schipani, G. Mazzola, L. Marty, M. Brescia, F. Cortecchia, F. Perrota e E. Rossi, Active optics control of the VST telescope with the CAN field-bus, Proceedings of ICALEPCS 2001 (International Conference on Accelerator and Large Experimental Physics Control System) Napoli, Italy, 2001.

[9] M. M. D. Santos e M. R. Stemmer, Uma rede CAN aplicada ao controle de uma aeronave não tripulada - O helicóptero hélix, XIV Congresso brasileiro de Automática, Natal, Brasil (2002).

[10] F.G. Moraes, A. M. Amory e J. P. Júnior, Sistema integrado e multiplataforma para controle remoto de residências, VII Workshop Iberchip, Montevidéu, Uruguai, 2001.

[11] M. A. Livani and J. Kaiser, EDF consensus on CAN bus access for dynamic real-time applications, IPPS/SPDP Workshops, 1998.

[12] Analog Device, Writing an ADuC812 Application in C, Microconverter Technical Note - uC002, Norwood, MA - USA, Inc. 2000.

[13] Texas Instruments. Getting Started in C and Assembly Code With the TMS320LF240x DSP - SPRA755A, USA, Inc, 2002.

[14] Texas Instruments. Code Composer User's Guide - SPRU296, USA, Inc, 1999.

[15] Texas Instruments. Quick Start Guide and Real-Timer Tutorial - SPRU510, USA, Inc, 2001.

[16] Texas Instruments. TMS320LF2407A DSP Controllers - SPRS145I, USA, Inc, 2003.

[17] Texas Instruments. TMS320F/C24x DSP Controllers CPU and Instruction Set Reference Guide - SPRU160C, USA, Inc, 1999.

[18] Texas Instruments. TMS320LF/LC240x DSP Controllers System and Peripherals Reference Guide - SPRU357B, USA, Inc, 2001.

[19] Lapsley, P.; Bier, J.; Shoham, A. e Lee, Edward A. DSP Processor Fundamentals: Architectures and Features, IEEE Press, 1997.

[20] Microchip, MCP2551 Data sheet, Microchip technology, USA, inc, 2002.

[21] Microchip, MCP2510 Data Sheet, Microchip technology, USA, inc, 2002.

[22] P. Richards, An in-depht look at the MCP2510 - AN 739, Microchip technology, USA, inc, 2001.

[23] Analog Device, ADuC812 User's Manual, MicroConverter Products, Norwood, MA - USA, Inc, 2000.

[24] Analog Device, MicroConverter, Multichannel 12-Bit with Embedded Flash MCU, MA - USA, Inc. 2000.

[25] H. Estl, SPI interface and use in a daisy-chain bus configuration, Infineon technologies, 2002.

Capítulo 6

ANEXO A - TMS320LF2407A

6.1 Introdução

O dispositivo TMS320LF2407A é um processador digital de sinais da família TMS320C2000 da *Texas Instruments*. Esse processador foi projetado para controlar motores digitais e outras aplicações de controle embarcado. Possui um CPU *core* baseado no processador C2xx de 16 *bits*, ponto fixo e de baixo consumo de potência. Utiliza uma arquitetura do tipo *Harvard*. Nesse tipo de arquitetura, o barramento da memória de dados e de programa são separados. Isto permite que os dados e as instruções sejam lidas simultaneamente, proporcionando uma alta velocidade na execução dos programas.

6.2 Características do TMS320LF2407A

O TMS320LF2407A possui as seguintes características:

- Ciclo de instrução de até 25 ns;
- Até 40 MIPS (Milhões de Instruções Por Segundo);
- CPU *core* baseado TMS320C2xx;
- Memória interna ao *chip*:
  - 32 K *Words* x 16 *Bits* de *Flash*/EEPROM;
  - 2,5 K *Words* x 16 *Bits* de RAM (dados/programa): 544 *Words* de DARAM e 2 K *Words* de SARAM;
  - Código de segurança para programação da memória *Flash*.

- *Boot* ROM;
- Dois módulos gerenciadores de evento (EVA e EVB). Cada um inclue:
  - Dois *timers* de 16 *bits*;
  - Oito canais PWM;
  - Seis unidades de comparação;
  - Três unidades de captura;
- *Interface* com memória externa:
- Módulo *Watchdog Timer*;
- Um conversor A/D de 10 *bits* com:
  - Dezesseis canais;
  - Tempo mínimo para conversão A/D de 500 ns.
- Módulo SCI (*Serial Communications Interface*);
- Módulo SPI (*Serial Peripheral Interface*)
- Módulo CAN (*Controller Area Network*);
- Entradas e saídas digitais (40 pinos);
- Cinco interrupções externas;
- Três modos de baixa potência;
- Emulador JTAG IEEE 1149.1.

Na figura 6.1 está ilustrado o diagrama de blocos do DSP TMS320LF2407A.

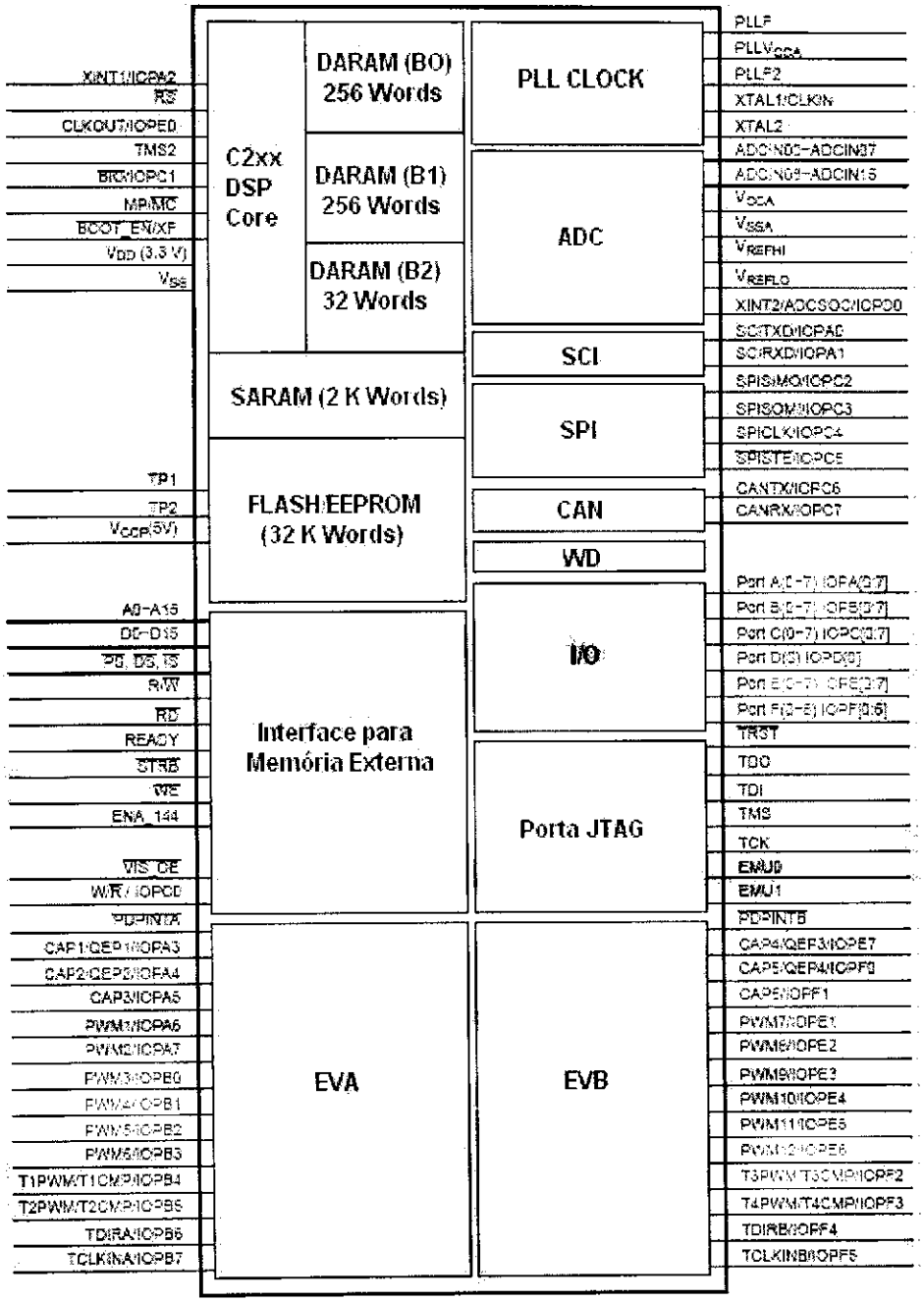

Figura 6.1: Diagrama de Blocos do TMS320LF2407A.

6.3 Arquitetura do Dispositivo

A *interface* entre os periféricos e a memória interna é realizada através da *interface* PBUS (Peripheral BUS). Os periféricos internos ao *chip* são acessados (leitura e escrita) em um único ciclo e com um estado de espera nulo. Todos os periféricos excluindo o *watchdog timer*, utilizam o *clock* da CPU. Na figura 6.2 está ilustrada a arquitetura do dispositivo.

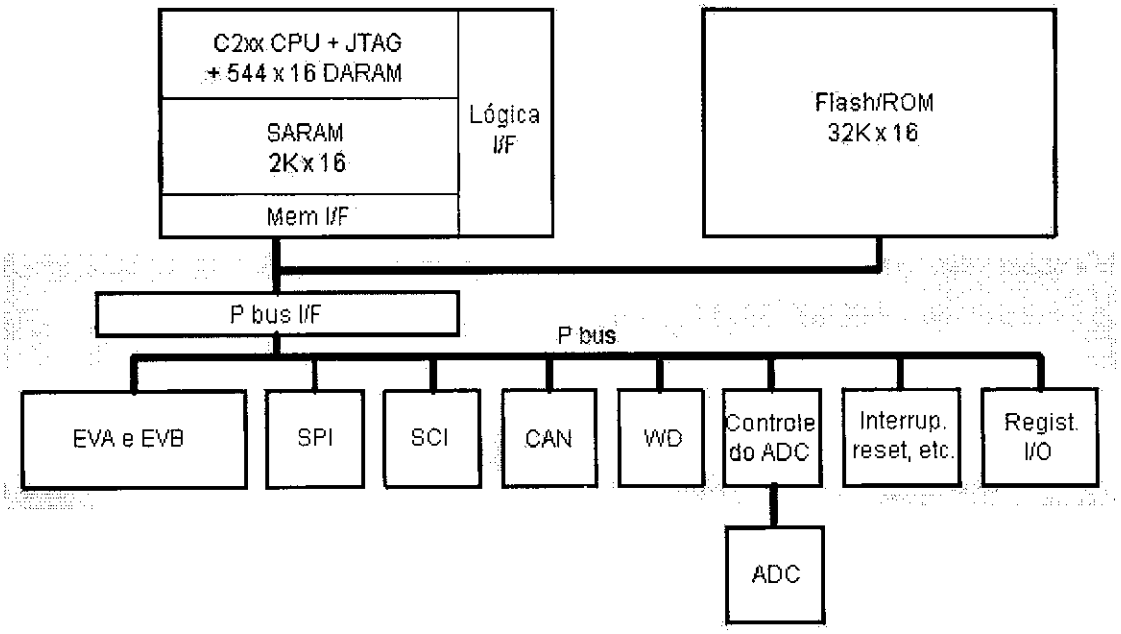

Figura 6.2: Diagrama de Blocos da Arquitetura do TMS320LF2407A.

6.4 Memória

O TMS320LF2407A possui um barramento de endereço de 16 *bits* que acessa os seguintes espaços, individualmente:

- Espaço de programa - 64 K*words*;
- Espaço de dados - 64 K*words*;
- Espaço de I/O - 64 K*words*;

Existem interno ao *chip* uma memória com acesso duplo (DARAM) e uma memória de programa/dados com acesso único (SARAM).

A DARAM intern ao *chip* de 544 x 16 *bits* pode ser acessada duas vezes por ciclo de máquina. As 544 *words* são divididas em três blocos: B0, B1 e B2. Esta memória é utilizada como memória de dados, mas o bloco B0 pode também ser usada como memória de programa. O bloco B0 pode ser configurado em um dos dois modos, dependendo do valor do *bit* **CNF** do registrador **ST1**.

Existem 2 K*words* x 16 *bits* de SARAM. Pode-se utilizar a SARAM como memória de dados e de programa.

A memória *Flash* é mapeada no espaço da memória de programa. O TMS320LF2407A possui 32 K *words* de memória Flash divididos em quatro setores (4K, 12K, 12K e 4K). Essa memória é utilizada para fornecer um armazenamento permanente do programa. Ela pode ser programada e apagada eletricamente várias vezes para permitir o desenvolvimento do código.

6.4.1 Memória de Programa

Além de armazenar o código do usuário, a memória de programa também armazena operandos imediatos e uma tabela de informações. Um máximo de 64 K*words* pode ser endereçado na memória de programa do TMS320LF2407A. Este número inclue a DARAM e Flash/EEPROM internas ao *chip*.

Sempre que uma locação e memória externa ao *chip* for acessada, os sinais de controle para acesso externo (/PS, /DS, /STRB, etc) serão automaticamente gerados. Na figura 6.3 está ilustrado o mapa da memória de programa.

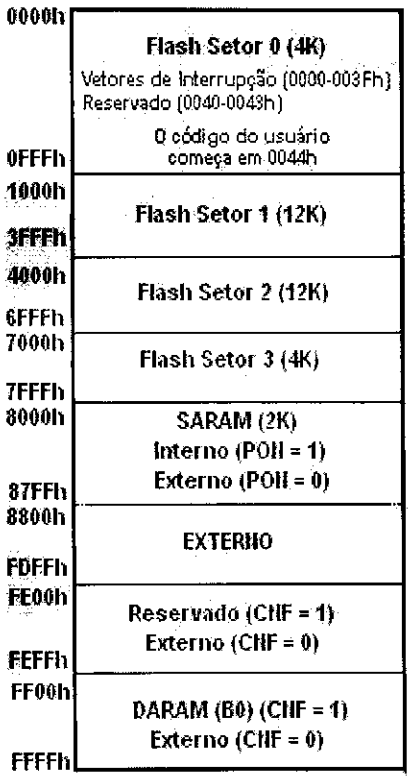

Figura 6.3: Esquema do Mapa da Memória de Programa.

6.4.2 Memória de Dados

Existe 64 K*words* de memória de dados. A metade desse espaço (0000 - 7FFF) é interna ao *chip*. A memória de dados interna inclui registradores de mapeamento da memória, DARAM e registradores de mapeamento da memória do periférico. Os 32 K*words* restantes de memória (8000 - FFFF) formam parte da memória de dados externa. Na figura 6.4 está ilustrado o mapa da memória de dados do TMS320LF2407A.

Figura 6.4: Esquema do Mapa da Memória de Dados.

6.4.3 Espaço I/O

Existem 64 K*words* no espaço I/O. Na figura 6.5 está ilustrado o mapa com os endereços do espaço I/O do TMS320LF2407A.

Figura 6.5: Esquema do Mapa com os Endereços do Espaço de I/O.

6.5 Clocks e Modos de Baixa Potência

6.5.1 Pinos Associados com o Clock

Existem três pinos no dispositivo que estão associados com os *clocks*:

- **XTAL1/CLKIN** - Este é o pino que contém o *clock* de entrada do cristal externo para o oscilador interno ao *chip*.
- **XTAL2** - Este é o pino que contém o *clock* de saída do oscilador interno ao *chip* para guiar o cristal externo.
- **CLKOUT/IOPE0** - Este é o pino de saída do *clock*. Ele é multiplexado com o pino 0 da porta E (IOPE0). Este pino pode ser usado como saída do *clock* da CPU ou do *clock* do *watchdog timer*, dependendo do valor do *bit* 14 do registrador **SCSR1**.

6.5.2 Phase Locked Loop - PLL

O TMS320LF2407A utiliza um circuito PLL embarcado no CPU *core*. O propósito de usar um PLL é multiplicar a frequência externa de referência (10 MHz) para uma frequência maior ou menor, que será usada internamente como *clock* da CPU.

Esse PLL dispõe de fatores de multiplicação que variam de 0,5 a 4 vezes a frequência do *clock* de entrada. Esse fator de multiplicação é configurado nos *bits* 11-9 do registrador **SCSR1**. Como a frequência do *clock* de entrada é de 10 MHz, é possível atingir uma frequência do *clock* da CPU de até 40 MHz, basta configurar o PLL para multiplicar por 4.

6.5.3 Watchdog timer Clock - WDCLK

Um clock de baixa frequência, **WDCLK**, é usado pelo *clock* do *watchdog timer*. **WDCLK** possui uma frequência nominal de 78125 Hz quando o *clock* da CPU (**CPUCLK ou CLKOUT**) for 40 MHz.

O WDCLK é gerado no periférico *watchdog timer*. A equação abaixo mostra a relação entre o WDCLK e o CLKOUT.

$$WDCLK = \frac{CLKOUT}{512}$$

6.5.4 Modos de Baixa Potência

O 2407A possui uma instrução **IDLE**, que quando executada, desliga os *clocks* de todos os circuitos da CPU. Entretanto, o *clock* de saída da CPU continua em atividade. Com esta instrução, os *clocks* da CPU são desligados para diminuir o consumo de energia. A CPU sai do estado **IDLE** se for resetado ou se receber uma requisição de interrupção.

Existem dois *bits* de controle que especificam qual modo de baixa potência será utilizado após a instrução **IDLE** ser executada. Esses *bits* de controle estão localizados nos bits 13-12 do registrador **SCSR1**.

Existem três modos de baixa potência: **IDLE1**, **IDLE2** e **HALT**. Quando a CPU entra no modo **IDLE1**, o *clock* da CPU é parado, mas o *clock* do sistema continua em atividade. O modo **IDLE2** é implementado por uma lógica externa. Nesse modo, o *clock* da CPU e do sistema são desligados. No terceiro modo de baixa potência, modo **HALT**, o **clock** de entrada para o PLL é desligado.

6.6 Entradas e Saídas Digitais - I/O

Existem seis portas no TMS320LF2407A: as portas A, B, C, D, E e F. As portas A, B, C e E possuem oito pinos, a porta D possui apenas um pino, e a porta F possui sete pinos. Cada pino de uma porta pode ser configurado para atuar como um pino de função I/O (função secundária), ou como um pino de função compartilhada (função primária).

Todos os pinos de funções I/O e funções compartilhadas são controlados através de nove registradores de 16 *bits*. Esses registradores são divididos em dois tipos:

- **Registradores de Controle MUX I/O (MCRx)** - Esse registrador é utilizado para escolher entre a função primária ou secundária.
- **Registradores de Controle de Dados e Direção (PxDATDIR)** - Esse registrador é utilizado para controlar os dados e a direção dos dados, quando os pinos estão configurados como uma função I/O (função secundária).

6.7 Gerenciador de Eventos

O módulo gerenciador de eventos (EV) fornece várias funções que são particularmente utilizadas em aplicações de controle de motores.

Existem dois gerenciadores de eventos: o EVA e o EVB. Eles são idênticos. Cada EV do TMS320LF2407A possui os seguintes blocos funcionais:

- Dois *timers*;
- Três unidades de comparação;
- Circuitos PWM, que incluem os circuitos SVPWM (Space Vector PWM - Modulação Vetorial), unidades de geração de banda morta e lógica de saída;
- Três unidades de captura;
- Circuitos QEP (*Quadrature Encoder Pulse*);
- Lógica de interrupção.

Nas figuras 6.6 e 6.7 estão ilustrados, respectivamente, os diagramas de blocos dos módulos EVA e EVB. Veja que os dois módulos são idênticos.

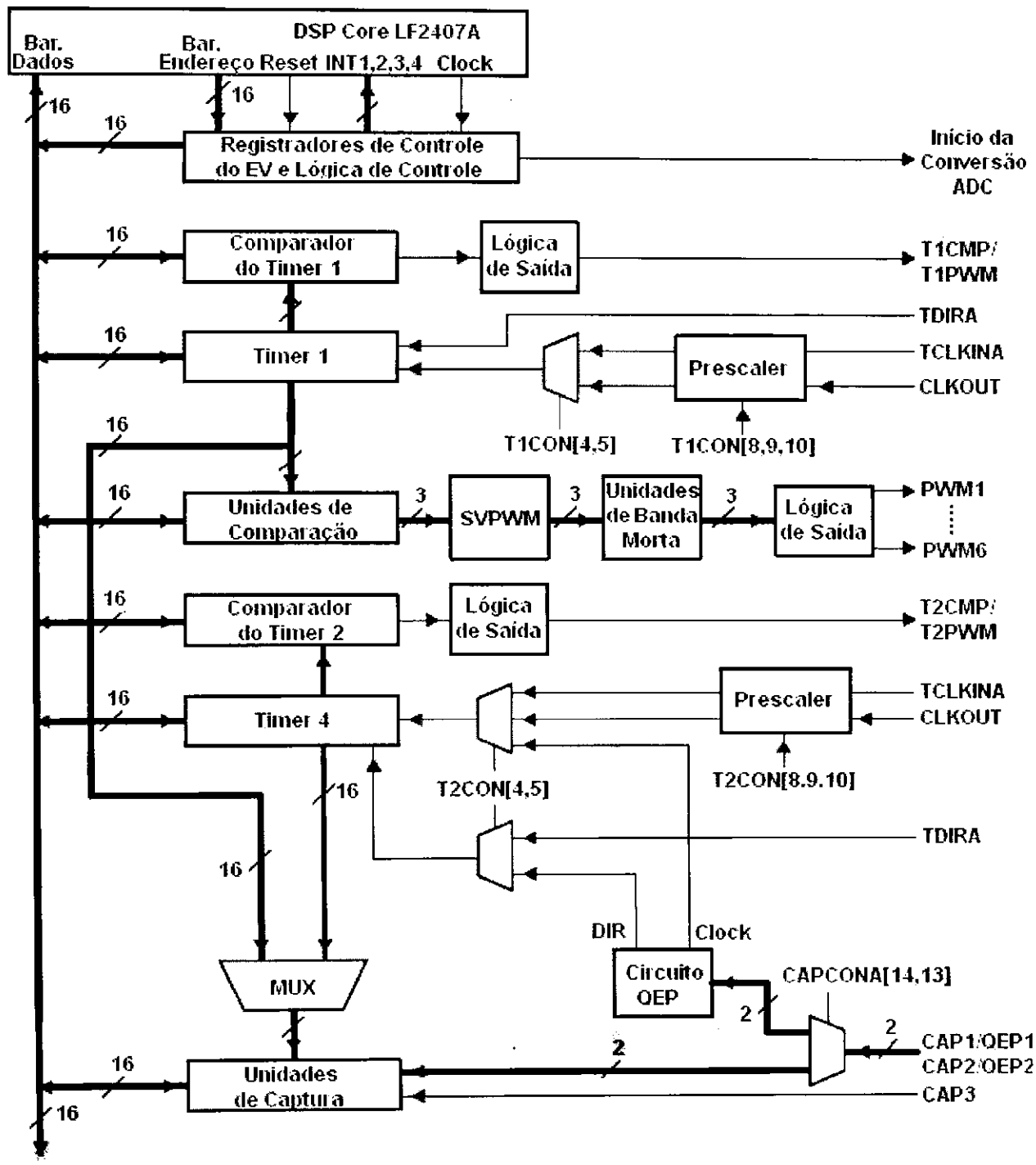

Figura 6.6: Diagrama de Blocos do Módulo EVA.

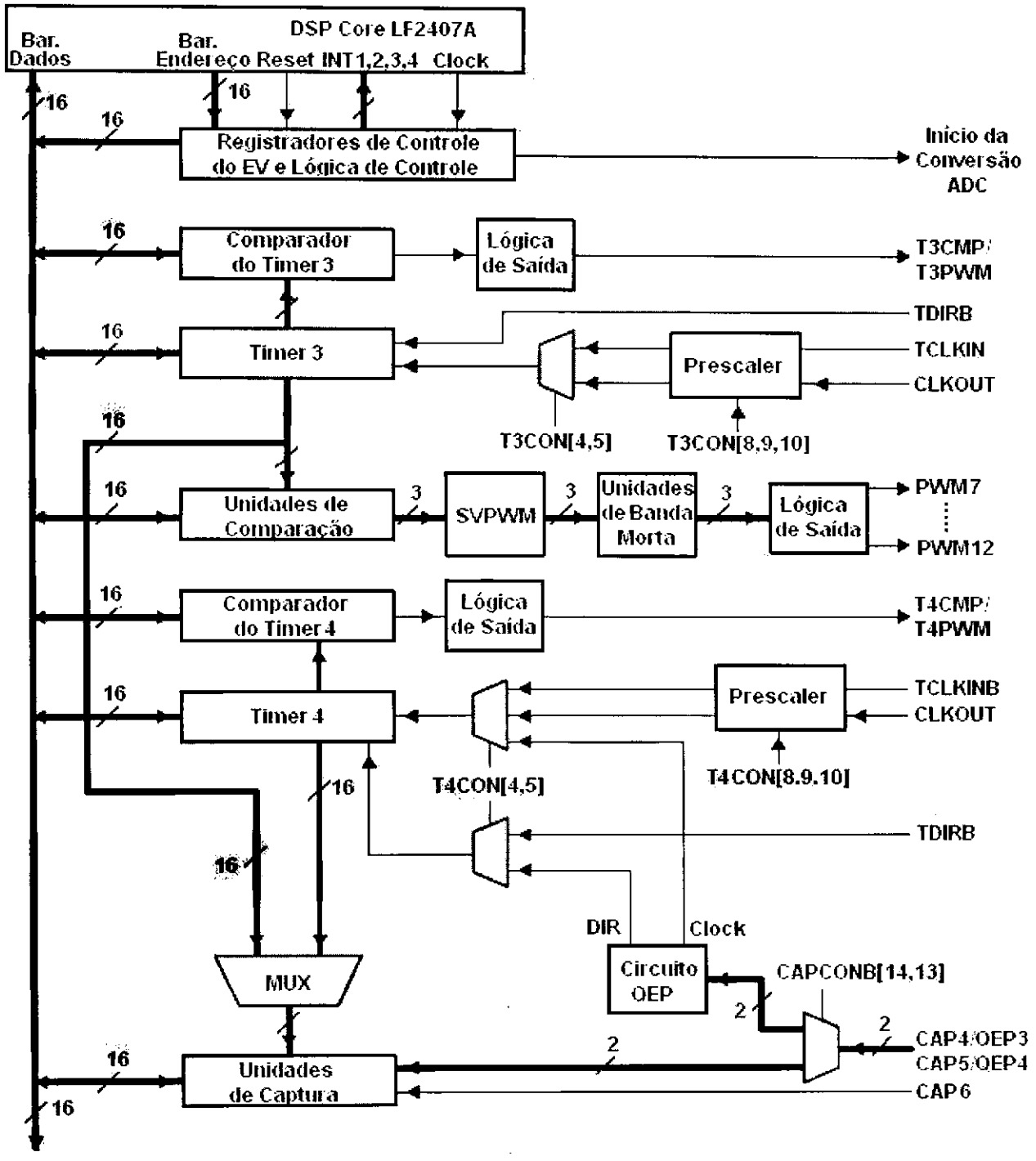

Figura 6.7: Diagrama de Blocos do Módulo EVB.

Cada módulo possui oito pinos disponíveis para as saídas de comparação/PWM, três pinos de entrada para captura e dois pinos de entrada do circuito QEP. Os *timers* podem ser programados para operar baseados em um *clock* externo ou interno ao dispositivo. O pino TCLKINA/B é a entrada do *clock* externo e o pino TDIRA/B é utilizado para especificar a direção de contagem, quando um timer está no modo de contagem crescente/decrescente direcional. Na tabela 6.1 estão indicados os pinos utilizados pelos módulos EVA e EVB e suas respectivas descrições.

Tabela 6.1: Descrição dos pinos utilizados pelos módulos EVA e EVB.

Pino - EVA	**Descrição**	**Pino - EVB**	**Descrição**
CAP1/QEP1	Unidade de Captura 1/ Entrada 1 do circuito QEP	CAP4/QEP3	Unidade de Captura 4/ Entrada 3 do circuito QEP
CAP2/QEP2	Unidade de Captura 2/ Entrada 2 do circuito QEP	CAP5/QEP4	Unidade de Captura 5/ Entrada 4 do circuito QEP
CAP3	Unidade de Captura 1	CAP6	Unidade de Captura 6
PWM1	Saída 1 da Unidade de Comparação 1	PWM7	Saída 1 da Unidade de Comparação 4
PWM2	Saída 2 da Unidade de Comparação 1	PWM8	Saída 2 da Unidade de Comparação 4
PWM3	Saída 1 da Unidade de Comparação 2	PWM9	Saída 1 da Unidade de Comparação 5
PWM4	Saída 2 da Unidade de Comparação 2	PWM10	Saída 2 da Unidade de Comparação 5
PWM5	Saída 1 da Unidade de Comparação 3	PWM11	Saída 1 da Unidade de Comparação 6
PWM6	Saída 2 da Unidade de Comparação 3	PWM12	Saída 2 da Unidade de Comparação 6

Continuação da tabela 1.2.

Pino - EVA	Descrição	Pino - EVB	Descrição
T1CMP/T1PWM	Saída de Comparação/PWM do Timer 1	T3CMP/T3PWM	Saída de Comparação/PWM do Timer 3
T2CMP/T2PWM	Saída de Comparação/PWM do Timer 2	T4CMP/T4PWM	Saída de Comparação/PWM do Timer 4
TCLKINA	Entrada do Clock Externo para os Timers do EVA	TCLKINB	Entrada do Clock Externo para os Timers do EVB
TDIRA	Entrada para Direcionar o Timer Externo no EVA	TDIRB	Entrada para Direcionar o Timer Externo no EVA

6.8 Conversor Analógico Digital

O conversor analógico digital do TMS320LF2407A possui as seguintes características:

- Um ADC de 10 *bits* com S/H (*Sample and Hold*);
- Tempo de conversão (S/H + conversão) de 375 ns;
- Dezesseis entradas analógicas multiplexadas (ADCIN0 - ADCIN15);
- Capacidade de autosequenciamento. Até 16 autoconversões em uma única sessão. Cada sessão pode ser programada para selecionar ao menos um dos dezesseis canais de entrada;
- Dois sequenciadores de oito estados independentes (SEQ1 e SEQ2) que podem ser operados individualmente no modo de sequenciamento dual, e um sequenciador de dezesseis estados (SEQ) que opera no modo de sequenciamento cascateado. A palavra "estados"indica o número de autoconversões que podem ser desempenhados com um sequenciador.
- Quatro registradores de controle do sequenciamento (**CHSELSEQn**) que determinam a sequência dos canais analógicos que são usados para conversão em algum modo de sequenciamento;
- Dezesseis registradores de resultados para armazenar os valores convertidos (**RESULT0 - RESULT15**);

- Várias formas de se iniciar uma conversão:
  - Por *software*: usando o *bit* SOC SEQn;
  - EVA: gerenciador de eventos A;
  - EVB: gerenciador de eventos B;
  - Externo: usando o pino ADCSOC.
- Flexível controle de interrupção;
- Sequenciador pode operar no modo *start/stop*, permitindo vários gatilhos de sequenciamento no tempo para sincronizar as conversões;
- EVA e EVB podem independentemente gatilhar, respectivamente, SEQ1 e SEQ2, apenas no modo de sequenciamento dual;
- Controle do *prescaler* para o tempo de aquisição da janela S/H;
- Modo de calibração.

6.9 Interface de Comunicação Serial - SCI

A *Interface* de Comunicação Serial (SCI) suporta a comunicação digital serial assíncrona UART (*Universal Asynchronous Receiver/Transmitter*) entre a CPU e outro periféricos assíncronos que usam o formato padrão NRZ (*Non-Return to Zero*).

O módulo SCI possui as seguintes características:

- Dois pinos de I/O: o pino de recepção de dados SCIRXD e o pino de transmissão de dados SCITXD;
- Taxa de transmissão programável;
- Comprimento da palavra de dados programável de um à oito *bits*;
- número de *bits* de stop programável em um ou dois;
- *Clock* gerado internamente;
- Quatro *flags* de detecção de erro;
- Dois modos *wake-up*;

- Operação *half-* ou *full - duplex*;
- Duplo armazenamento nas funções de transmissão e recepção;
- Um transmissor e um receptor podem ser operados por interrupções ou por *polling* do *status* dos *flags*;
- Formato NRZ.

Na figura 6.8 está ilustrado o diagrama de blocos do módulo SCI. Observe que todos os registradores são de oito *bits*.

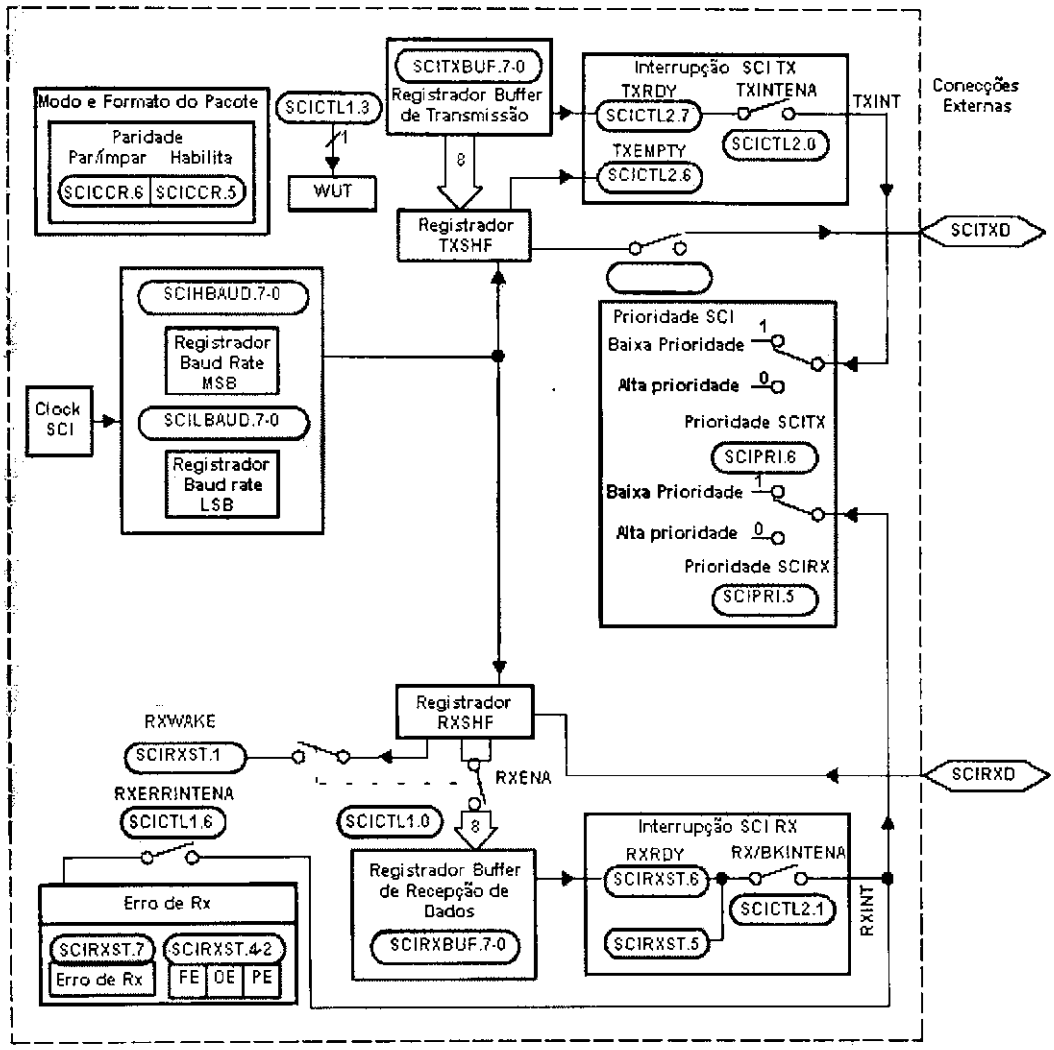

Figura 6.8: Diagrama de Blocos do Módulo SCI.

6.10 Interface Periférica Serial (SPI)

A *interface* SPI é uma porta de I/O serial síncrona de alta velocidade que permite a transmissão de dados (1 à 16 *bits*) entre um dispositivo mestre e um ou vários dispositivos escravos.

O módulo SPI consiste de:

- Quatro pinos I/O: SPISIMO, SPISOMI, SPICLK e SPISTE;
- Modos de operação mestre e escravo;
- Registrador *buffer* de recepção serial **(SPIRXBUF)**;
- Registrador *buffer* de transmissão serial **(SPITXBUF)**;
- Registrador de dados **(SPIDAT)**;
- Controle de polaridade e fase do SPICLK (*clock* da SPI);
- Lógica de controle;
- Registradores de *status* e controle mapeados na memória.

Existem quatro pinos I/O na *interface* SPI: o SPISIMO, SPISOMI, SPICLK e o SPISTE.

O **SPISIMO** (*SPI Slave In Master Out*) é um pino de saída no dispositivo mestre e um pino de entrada no dispositivo escravo. Os dados são transferidos do dispositivo mestre para o dispositivo escravo por esse pino.

O **SPISOMI** (*SPI Slave Out Master In*) é um pino de saída no dispositivo escravo e um pino de entrada no dispositivo mestre. Os dados são transferidos do dispositivo escravo para o dispositivo mestre por esse pino.

O **SPICLK** (*SPI Clock*) é um pino que contém o *clock* da SPI. No dispositivo mestre esse pino é usado como um pino de saída e no dispositivo escravo esse pino é usado como um pino de entrada. Portanto, o dispositivo mestre controla a transferência dos dados. A frequência do *clock* não poderá ser maior que 1/4 do *clock* do dispositivo (CLKOUT).

O **SPISTE** (*SPI Select Slave*) é um pino de entrada no dispostivo escravo. Esse pino é ativo em nível lógico baixo. A transferência de dados será realizada somente se o nível lógico nesse pino for baixo.

Na figura 6.9 está ilustrado um diagrama de blocos do módulo SPI.

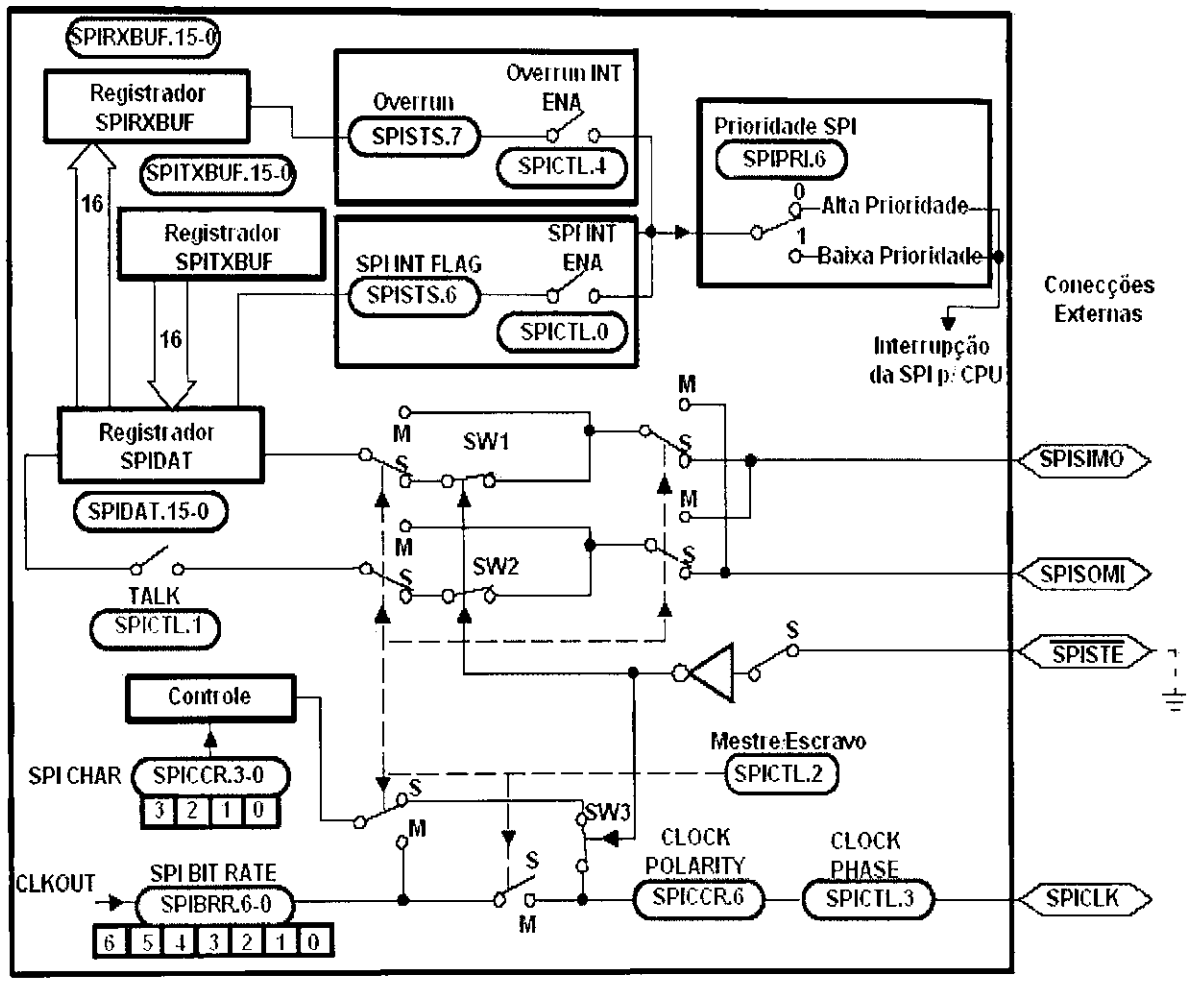

Figura 6.9: Diagrama de Blocos do Módulo SPI.

6.11 Módulo CAN

O periférico CAN disponível no TMS320LF2407A possui as seguintes características:

- Implementação do protocolo CAN, versão 2.0B;
- Seis *mailbox* (MBOX) com comprimento de dados de 0 à 8 *bytes*;
- Duas máscaras de aceitação local;
- Taxa de *bits* programável;
- Esquema de interrupção programável;
- *Wake-up* programável;
- Resposta automtica a uma requisição remota;
- Re-transmissão automática no caso de erros ou preca de arbitragem;
- Diagnóstico de falhas no barramento;
- Modo *self-test*;

O *mailbox* (MBOX) é a área onde os pacotes CAN são armazenados antes de serem transmitidos, e onde são recebidos. Os MBOX 0 e 1 são *mailbox* de recepção, os MBOX 4 e 5 são mailbox de transmissão e os MBOX 2 e 3 podem ser configurados como MBOX de transmissão ou de recepção. Cada *mailbox* possui quatro registradores de 16 *bits*, no qual podem armazenar um máximo de oito *bytes*. Na figura 6.10 está ilustrado o diagrama de blocos do módulo CAN do TMS320LF2407A.

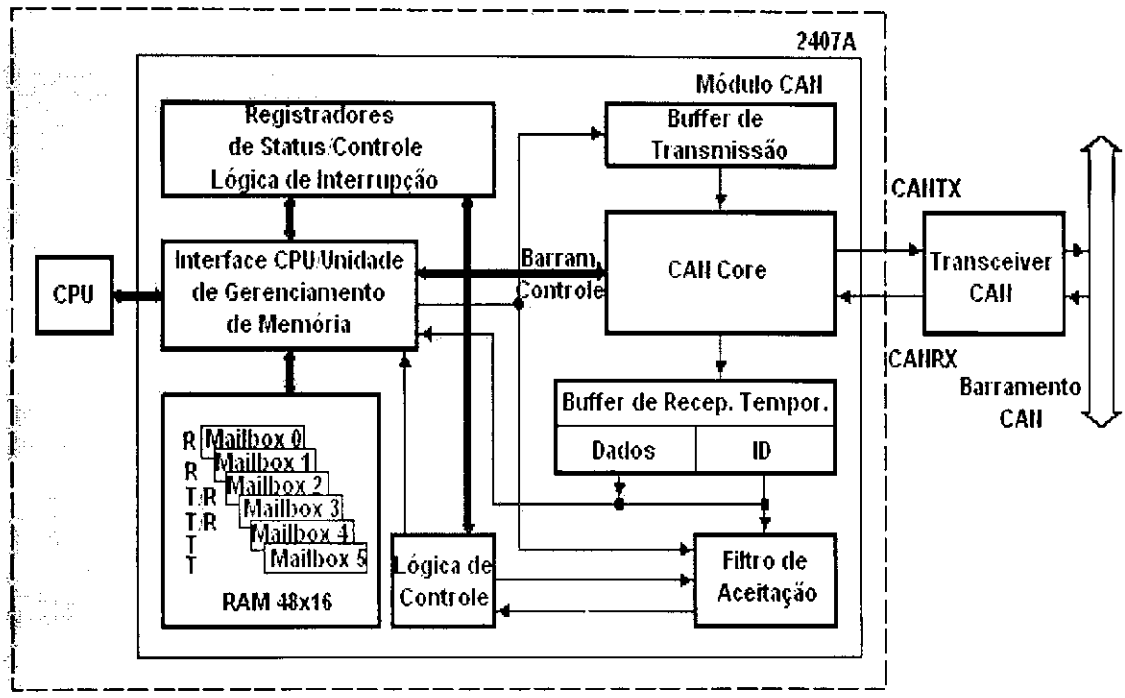

Figura 6.10: Diagrama de Blocos do Módulo CAN do TMS320LF2407A.

6.12 Watchdog Timer

O *watchdog* (WD) *timer* monitora as operações do *software* e do *hardware* e implementa funções de *reset* do sistema. Se o *software* entrar em um *loop* impróprio ou se a CPU se "perder"temporariamente, um *overflow* do *watchdog timer* assegurará um *reset* do sistema. O WD *timer* aumenta a confiabilidade da CPU e deste modo assegura a integridade do sistema.

O módulo *watchdog timer* possui as seguintes características:

- O contador do WD de oito *bits* gera um *reset* do sistema quando ocorrer um *overflow*;
- O Contador *free-running* de seis *bits* que alimenta o contador do WD via o *prescaler* do contador do WD;
- Um registrador chave de *reset* **(WDKEY)** que zera o contador do WD quando a correta combinação de valores for escrita, e gera um *reset* se um incorreto valor for escrito no registrador;

- O WD verifica os *bits* que iniciam um *reset* do sistema se o WD for corrompido;
- Ativação automatica do WD *timer* uma vez que o *reset* do sistema foi liberado;
- Um *prescaler* do WD com seis seleções do contador *free-running* de seis *bits*.

Na figura 6.11 está ilustrado o diagrama de blocos do módulo WD.

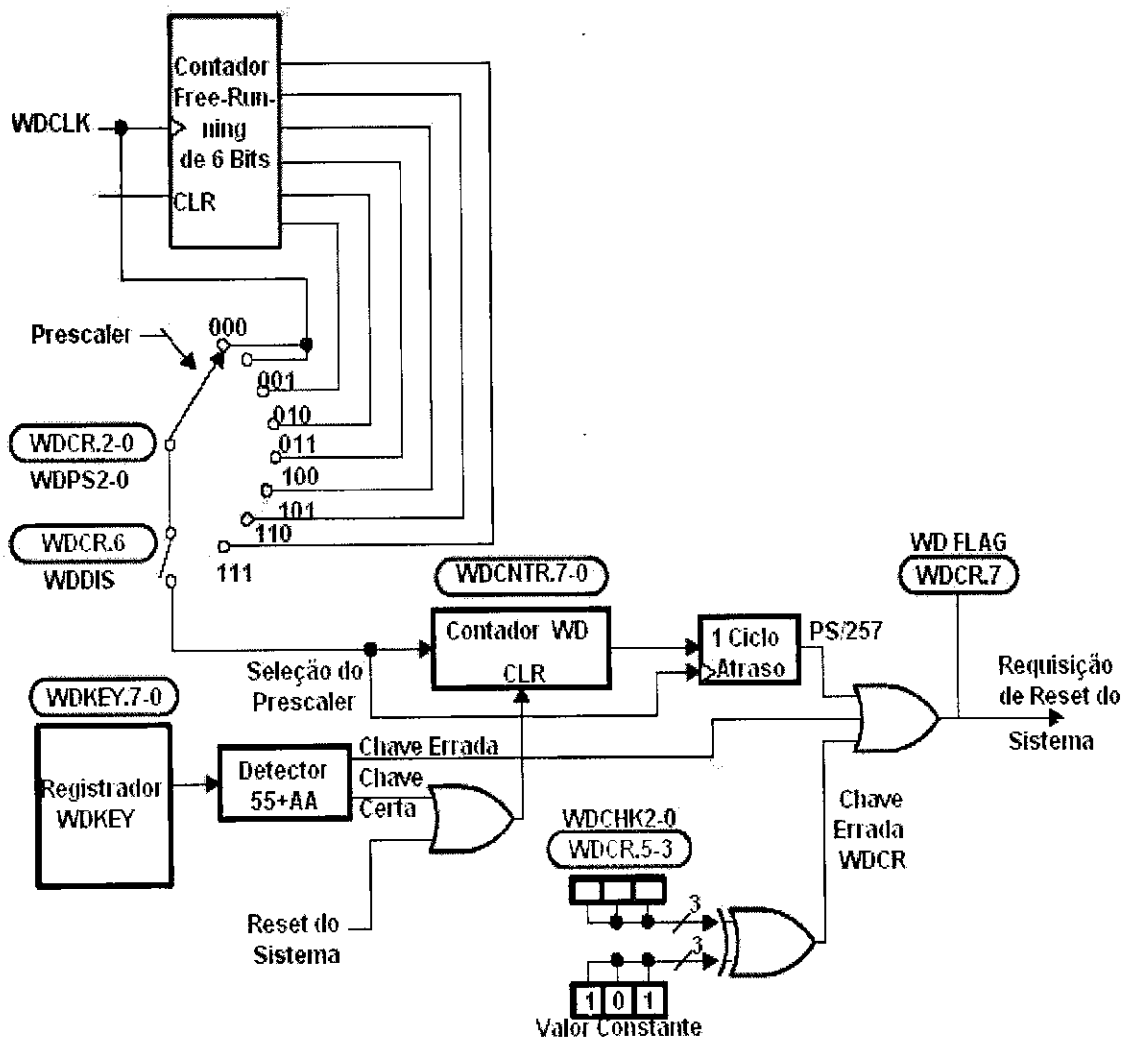

Figura 6.11: Diagrama de Blocos do Módulo WD.

Capítulo 7

ANEXO B - ADuC812

7.1 Descrição Geral

O microconversor ADuC812 é um circuito integrado com um sistema de aquisição de dados de 12 *bits* de alto desempenho, incorporando um conversor multicanal ADC, dois conversores DAC de 12 *bits* e um processador de 8 *bits* (compatível com o conjunto de instruções do 8051), todos incluídos em um único *chip*.

O processador 8051 possui uma memória de programa *Flash*/EE de 8k *bytes*, uma memória de dados *Flash*/EE de 640 *bytes* e uma memória de dados SRAM de 256 *bytes*. Possui 32 linhas programáveis de E/S, uma *interface* SPI, uma porta de E/S serial padrão UART e três temporizadores/contadores de 16 *bits*. O ADuC812 usa um *clock* interno de 11.0592 MHz. Na Figura 7.1 está ilustrado o diagrama de bloco funcional do microconversor ADuC812.

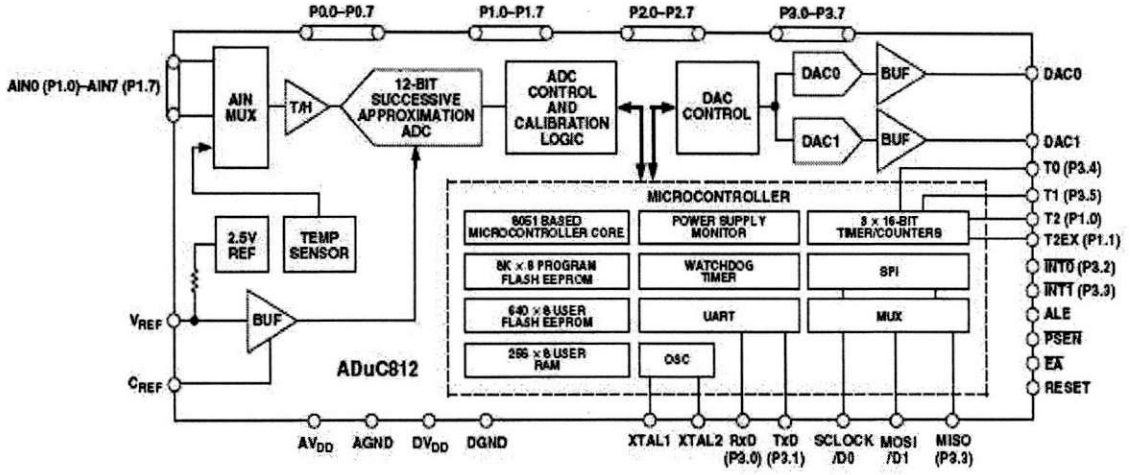

Figura 7.1: Diagrama de Blocos do Microconversor ADuC812.

7.2 Organização da Memória

O ADuC812 possui separado os espaços de endereço da memória de dados e de programa, como ilustrado nas figuras 7.2 e 7.3.

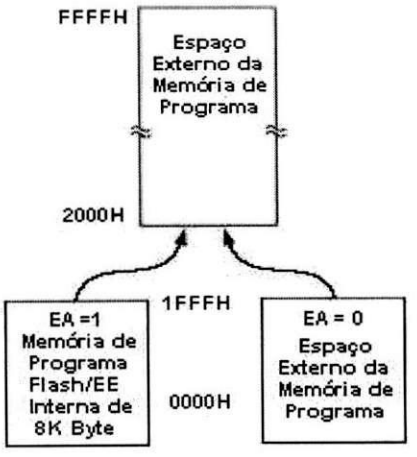

Figura 7.2: Diagrama com o Mapa do Espaço da Memória de Programa.

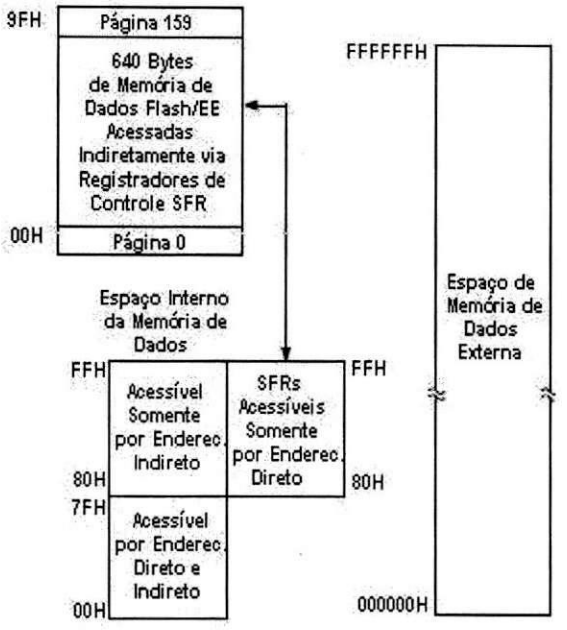

Figura 7.3: Diagrama com o Mapa do Espaço da Memória de Dados.

Na figura 7.3 está ilustrado os 640 *bytes* de memória de dados EEPROM/*Flash* que estão disponíveis ao projetista. A área de memória de dados *flash* é acessada indiretamente via um grupo de registradores que estão mapeados na área de registradores de funções especiais (SFR) no espaço de memória de dados.

O espaço dos registradores de função especial é mapeado em 128 *bytes* do espaço de memória de dados interna. A área SFR é acessada apenas por endereçamento direto, e fornece uma *interface* entre a CPU e todos os periféricos do *chip*. Na figura 7.4 está ilustrado um diagrama de blocos do modelo de programação do ADuC812 via a área SFR.

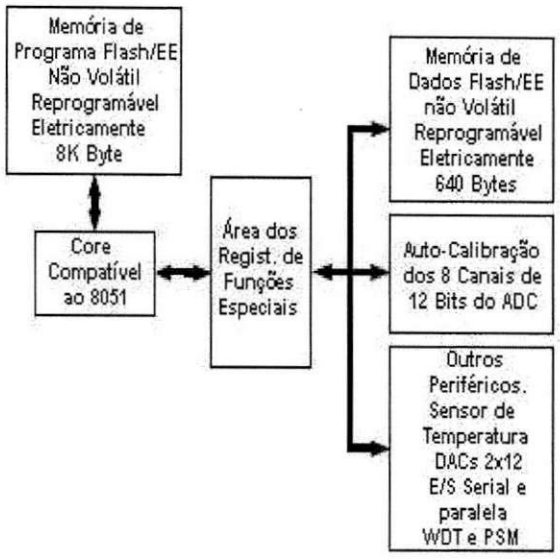

Figura 7.4: Diagrama de Blocos do Modelo de Programação do ADuC812 Via a Área SFR.

7.3 Portas de I/O

O ADuC812 utiliza 4 portas de entrada e saída para trocar dados com outros dispositivos externos. Algumas portas são capazes de efetuar operações de memória externa e outras são multiplexadas com uma função alternativa dos periféricos do dispositivo. Em geral, quando um periférico está habilitado, este pino pode não ser usado como um pino de propósito geral de entrada e saída.

7.4 *Interface* Serial UART

A porta serial possui capacidade *full duplex*, isto significa que é possível transmitir e receber dados simultaneamente. Também é possível receber um segundo dado antes de ler o primeiro dado que está armazenado no registrador de recepção (SBUF). Entretanto, se o primeiro *byte* não for lido em um certo intervalo de tempo após a recepção do segundo dado, o primeiro dado será perdido.

A *interface* física para a rede de dados serial é feita pelos pinos RXD (P3.0) e TXD (P3.1). A *interface* serial UART utiliza dois registradores SFR, o SBUF e o SCON. Os registradores de transmissão e recepção da porta serial são ambos acessados através do registrador SBUF. Para configurar e controlar a porta serial UART, utiliza-se o registrador SCON.

É possível configurar a UART em quatro diferentes modos:

- **Modo 0 (Registrador de Deslocamento de 8 *bits*)** - Nesse modo 8 *bits* de dados são transmitidos ou recebidos. A transmissão é iniciada quando se escreve no registrador SBUF. Os 8 *bits* são transmitidos, iniciado-se pelo *bit* menos significativo. A recepção é iniciada quando o *bit* de habilitação de recepção (REN) estiver em nível alto e o *bit* de interrupção de recepção (RI) estiver em nível baixo.

- **Modo 1 - Transmissão de 8 *Bits* com Taxa Variável** - Nesse modo cada *byte* é precedido por um *bit* de início (0) e seguido por um *bit* de parada (1). Na figura 7.5 está ilustrado esse processo. Portanto, 10 *bits* são transmitidos pelo pino TXD e conseqüentemente recebidos pelo pino RXD. A taxa de transmissão é ajustada pelo *timer* 1 ou 2 ou uma combinação dos dois (um para transmissão e o outro para a recepcão). A transmissão é iniciada quando se escreve no registrador SBUF e a recepcão é iniciada quando uma transição de 1 para 0 for detectada no pino RXD.

- **Modo 2 - Transmissão de 9 *Bits* com Taxa Fixa** - Nesse modo a taxa de transmissão é fixada em CLK_*Core*64 por *default* ou em CLK_Core32 e 11 bits são transmitidos e recebidos, um bit de início (0), 8 *bits* de dados, um nono *bit* programável e um *bit* de parada (1). O nono *u*tilizado como um *bit* de paridade. Para transmitir, os 8 *bits* de dados devem ser escritos no registrador SBUF. O nono *bit* deve ser escrito no *bit* TB8 de SCON. Quando a transmissão for iniciada os 8 *bits* de dados serão carregados no registrador de deslocamento de transmissão. O *flag* TI será setado assim que o *bit* de parada aparecer no pino TXD. Na recepção os 8 *bits* de dados são carregados no registrador de deslocamento de recepção.

- **Modo 3 - Transmissão de 9 *Bits* com Taxa Variável** Nesse modo a taxa de transmissão é determinada pelo *timer* 1 ou 2. Esse modo é semelhante ao modo 2, mas a taxa de transmissão pode ser variada como no modo 1.

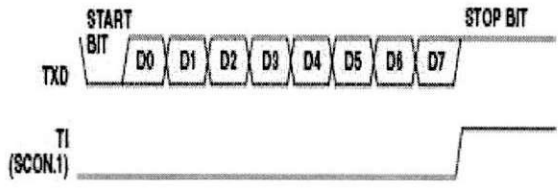

Figura 7.5: Processo de Transmissão de *Bits* Utilizando o Modo 1.

7.5 *Interface* SPI

O ADuC812 integra um *hardware* completo da *interface* SPI em um *chip*. A SPI é uma *interface* serial síncrona na qual dados de 8 *bits* podem ser transferidos de um ponto para outro a cada instante. Esta *interface* pode ser utilizada para se comunicar com um dispositivo periférico serial ou com outro microcontrolador que possua uma *interface* SPI. Os dados são transmitidos sincronamente e recebidos simultaneamente no modo *full duplex*. A porta SPI pode ser configurada para funcionar como mestre ou escravo e tipicamente consiste de quatro pinos: MISO (*Master In, Slave Out*), MOSI (*Master Out, Slave In*), SCLOCK (Serial *Clock*) e SS (*Slave Select*). Na figura 7.6 está ilustrado a conecção dos pinos da SPI entre um dispositivo mestre e um dispositivo escravo.

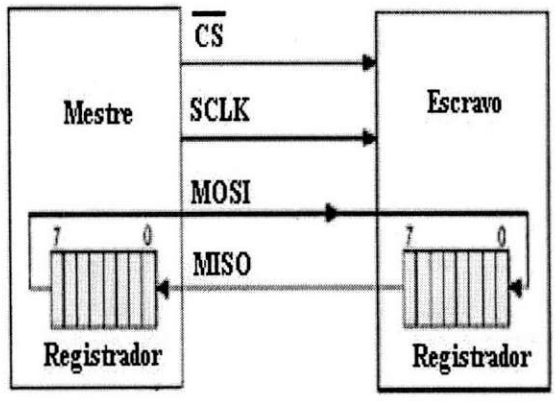

Figura 7.6: Esquema de Conecção dos Pinos entre um Dispositivo Mestre e um Escravo.

7.5.1 Pino MISO

O pino MISO é configurado como uma linha de entrada no modo mestre e como uma linha de saída no modo escravo. A linha MISO no mestre (entrada de dados) pode ser conectada na linha MISO no escravo (saída de dados). Os dados são transferidos como um dado serial de 8 *bits*.

7.5.2 Pino MOSI

O pino MOSI é configurado como uma linha de saída no modo mestre e como uma linha de entrada no modo escravo. A linha MOSI no mestre (saída de dados) deve ser conectada na linha MOSI do escravo (entrada de dados). Os dados são transferidos como um dado serial de 8 *bits*.

7.5.3 Pino SCLOCK

O *clock* é usado para sincronizar os dados transmitidos e recebidos pelas linhas de dados MOSI e MISO. Um único *bit* de dados é transmitido e recebido em cada período do SCLOCK. Portanto, um *byte* será transmitido/recebido após oito períodos do SCLOCK. O pino SCLOCK é configurado como uma saída no modo mestre e como uma entrada no modo escravo. No modo mestre a taxa de *bit*, a polaridade e a fase do *clock* são controladas pelos *bits* CPOL, CPHA, SPR0 e SPR1 no registrador de controle SPICON.

7.5.4 Pino SS

Esta linha é chamada em alguns dispositivos de CS - *Chip Select*, ela é ativa em nível baixo. Os dados são recebidos ou transmitidos no modo escravo quando o pino SS está em nível baixo permitindo que o ADuC812 seja usado como uma configuração de um único mestre e múltiplos escravos.

7.5.5 *Interface* SPI - Configuração Mestre

Na configuração mestre uma transmissão ou recepção é inicializada pela escrita no registrador SPIDAT. Oito períodos de *clock* são gerados pelo pino SCLOCK e o *byte* do registrador SPIDAT é transmitido pelo pino MOSI. A cada período do SCLOCK um *bit* de dados é amostrado pelo pino MISO. Após oito pulsos de *clock* o *byte* terá sido completamente transmitido e ficará esperando pelo registrador de deslocamento de entrada. O sinalizador ISPI será habilitado automaticamente e ocorrerá uma interrupção se esta estiver habilitada. O valor do registrador de deslocamento será armazenado no registrador SPIDAT.

7.5.6 *Interface* SPI - Configuração Escravo

Na configuração escravo o SCLOCK é uma entrada. O pino SS deve ser controlado externamente por um nível baixo durante a comunicação. A transmissão também é iniciada pela escrita no registrador SPIDAT. Nessa configuração, um *bit* de dados é transmitido por meio do pino MISO e um *bit* de dados é recebido por meio do pino MOSI através de cada entrada do período SCLOCK. Após oito pulsos de *clock*, o *byte* terá sido transmitido completamente e estará esperando na entrada do registrador de deslocamento. O sinalizador ISPI será habilitado automaticamente e ocorrerá uma interrupção se esta estiver habilitada. O valor no registrador de deslocamento será armazenado no SPIDAT apenas quando a transmisão/recepção de um *byte* estiver concluída. O fim da transmissão ocorrerá após oito pulsos de *clock* terem sido recebidos caso CPHA = 1 ou se SS retornar ao nível alto se CPHA = 0.

7.5.7 Modos de Operação

Dependendo da configuração dos bits do registrador de controle da *interface* SPI (SPICON), o ADuC812 transmitirá ou receberá dados em vários modos. Na figura 7.7 está ilustrada todas as possíveis configurações da SPI e as relações temporais e de sincronização entre os sinais envolvidos. Também é ilustrado o comportamento do bit de sinalização (ISPI) da *interface* SPI.

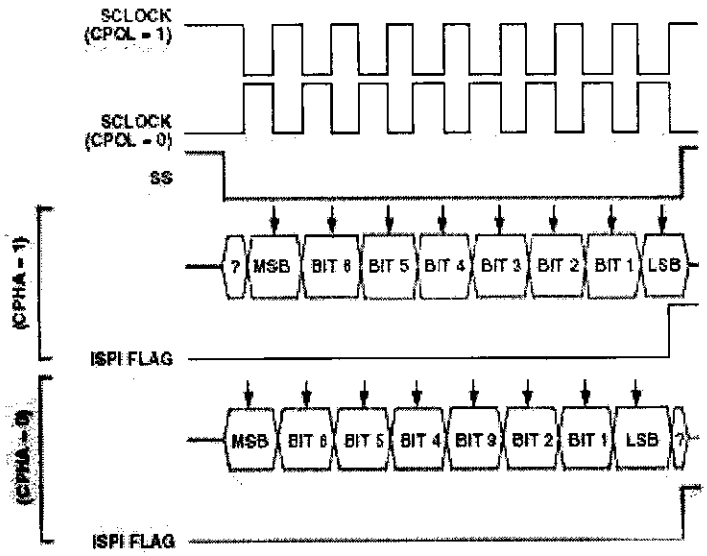

Figura 7.7: Esquema dos Sinais nos Pinos da SPI nos Vários Modos de Operação.

7.6 Módulos A/D e D/A

O módulo A/D incorpora um rápido conversor A/D de 8 canais e de 12 *bits*. Todos os componentes desse módulo são facilmente configurados utilizando três registradores SFR, o ADCCON1, ADCCON2 e o ADCCON3.

O conversor consegue converter um sinal na faixa de 0V à V_Ref. Uma vez configurada o ADC (*Analog Digital Converter*) converterá o sinal de entrada analógico e fornecerá um resultado em uma palavra de 12 *bits* que será armazenada nos registradores SFRs ADCDATAH e ADCDATAL. Os 4 *bits* mais significativos do ADCDATAH indicarão o canal de conversão que está sendo utilizado e os 4 *bits* restantes, estarão os 4 *bits* mais significativos do sinal convertido. No ADCDATAL, teremos os 8 *bits* restantes do sinal convertido. Na figura 7.8 está ilustrada o formato da palavra de 12 *bits* convertida no ADC.

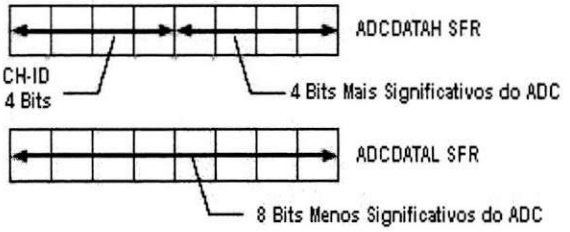

Figura 7.8: Formato do Sinal Resultante Convertido no ADC.

O ADuC812 possui dois DACs (*Digital Analog Converter*) de 12 *bits*. Cada DA pode operar no modo de 12 ou 8 *bits*. Cada um pode selecionar duas faixas, de 0V à V_Ref (2.5 V) ou de 0V à AV_DD(3V à 5V). Ambos compartilham um registrador de controle (DACCON) e quatro registradores de dados (DAC1H, DAC1L, DAC0H e DAC0L).

7.7 Temporizadores

O ADuC812 possui três temporizadores/contadores de 16 *bits*: *Temporizador 0*, *Temporizador 1* e o *Temporizador 2*. Cada um deles consistem de dois registradores de 8 *bits* que são THx e TLx (x = 0, 1 e 2), todos eles podem ser configurados para operar como temporizadores ou contadores de eventos.

Operando como temporizador, o registrador TLx é incrementado a cada ciclo de máquina. Como um ciclo de máquina consiste de 12 períodos de *clock* do *core* (12MHz), a máxima taxa será de 1/12 da frequência de *clock* do *core*.

Operando como contador, o registrador TLx é incrementado na transição de 1 para 0 no seu pino de entrada externo correspondente T0, T1 ou T2. Nessa função a entrada externa é amostrada durante cada ciclo de máquina. Quando as amostragens exibem um nível alto em um ciclo e um nível baixo no próximo ciclo, a contagem é incrementada. O novo valor de contagem aparecerá no registrador no novo ciclo de máquina. Como levam dois ciclos de máquina (24 períodos de *clock* do *core*) para reconhecer a transição de 1 para 0, a máxima taxa de contagem será de 1/24 da frequência de *clock* do *core*. Não há restrições sobre o ciclo de trabalho do sinal de entrada externo, mas para assegurar que uma dado nível será amostrado pelo menos uma vez antes que ele mude de valor, ele pode ser mantido por no máximo um ciclo de máquina completo.

Os temporizadores/contadores 0 e 1 possuem quatro modos de operação, e o temporizador/contador 2 possui dois modos de operação. A seguir veremos esses modos de operação.

Capítulo 8

ANEXO C - Controlador CAN MCP2510

8.1 Introdução

O MCP2510 é um controlador CAN completo com a arquitetura *Stand-Alone*, que implementa as especificações CAN 2.0A e 2.0B. Ele é capaz de transmitir e receber mensagens nos formatos padrão e estendido a taxas de até 1Mbps. Possui 6 filtros e 2 máscaras de aceitação, que trabalham em conjunto com três *buffers* de transmissão e dois *buffers* de recepção.

O MCP possui a *interface* SPI (*Serial Peripheral Interface*) programado para operar nos modos 0,0 e 1,1. Com essa *interface* é possível se comunicar com um outro dispositivo, com taxas de dados que podem alcançar 5 Mbps.

O dispositivo consiste de três blocos principais:

- **Protocolo CAN** - Implementa o protocolo CAN. Ele gerencia toda parte de transmissão e recepção de mensagens através do barramento.
- **Controle Lógico** - É utilizado na configuração e operação do dispositivo.
- **Protocolo SPI** - É responsável pelo funcionamento da *interface* SPI.

Nas figuras 8.1 e 8.2 estão ilustradas, respectivamente, a pinagem e o diagrama de blocos do controlador CAN MCP2510.

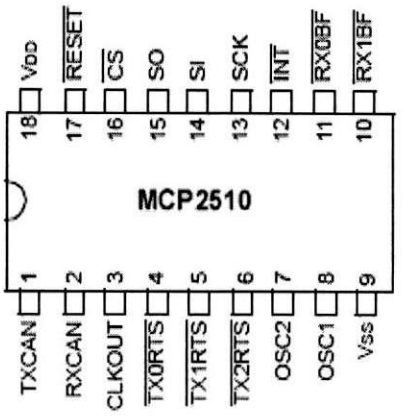

Figura 8.1: Esquema da Pinagem do MCP2510

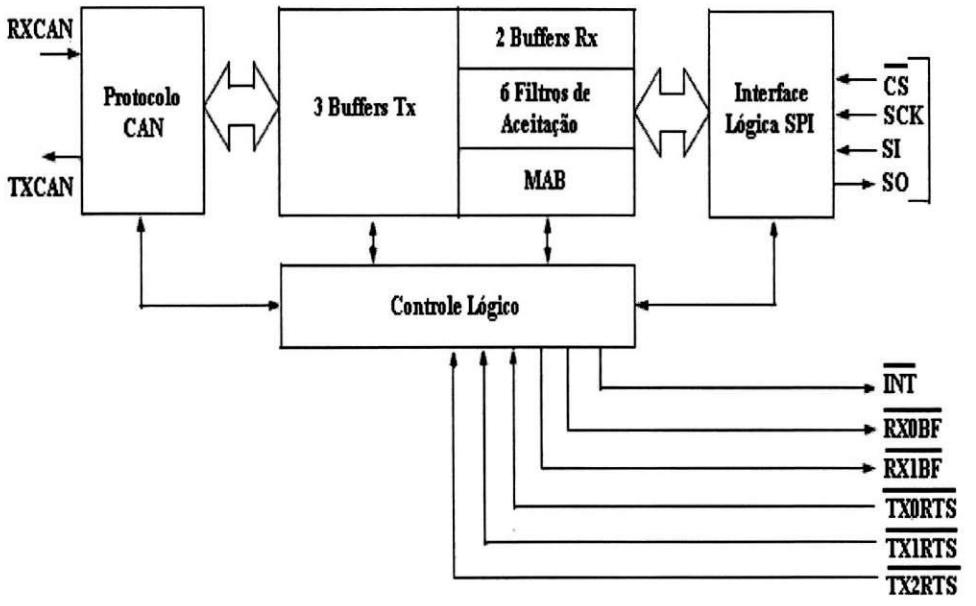

Figura 8.2: Diagrama de Blocos do MCP2510

8.2 Transmissão de Mensagens

Existem três *buffers* de transmissão. Cada *buffer* ocupa 14 *bytes* de memória do dispositivo. O primeiro *byte* é formado por um registrador de controle associado com o *buffer* de transmissão de mensagem. A informação neste registrador determina as condições sobre as quais a mensagem será transmitida e indica o *status* da transmissão da mensagem.

Cinco *bytes* são usados para manter os identificadores padrão e estendido e outras informações de arbitração e de controle da mensagem. Os últimos 8 *bytes* formam o campo de dados do pacote. Eles são utilizados para os possíveis 8 *bytes* de dados da mensagem a ser transmitida.

8.2.1 Prioridade de Transmissão

O MCP2510 tem a capacidade de atribuir prioridades aos *buffers* de transmissão. Tal prioridade não tem relação alguma com a priorização do protocolo CAN quando ocorre a arbitragem pelo controle do barramento. Antes de enviar o *bit* SOF (*Start Of Frame*), a prioridade de todos os *buffers* que estão enfileirados para a transmissão é comparada. O *buffer* de transmissão com maior prioridade será enviado primeiro.

8.2.2 Iniciando a Transmissão

Para iniciar a transmissão da mensagem um *bit* chamado TXREQ do registrador de controle do respectivo *buffer* deverá ser setado. Isto pode ser feito escrevendo no registrador via *interface* SPI ou setando o respectivo pino /TXnRTS. O fato de setar este *bit*, não implica em um início de transmissão da mensagem, isto somente indica que esse *buffer* está carregado. A transmissão só começará quando o dispositivo detectar que o barramento está livre.

Se a transmissão for completada com sucesso, o *bit* TXREQ será zerado e uma interrupção será gerada caso o projetista tenha habilitado a correspondente interrupção. Caso a transmissão da mensagem falhe, o *bit* TXREQ permanecerá setado indicando que a mensagem ainda está pendente para transmissão.

8.2.3 Pinos TXnRTS

São pinos que podem ser configurados como entradas para requisitar o envio de uma mensagem ou como entradas digitais padrão.

A configuração e controle desses pinos é realizada através do registrador TXRTSCTRL. Esse registrador só pode ser modificado quando o MCP2510 estiver no modo de configuração.

8.2.4 Abortando uma Transmissão

O MCU pode requisitar um aborto de mensagem em um *buffer* de mensagem específico, zerando o *bit* TXREQ. É possível também, abortar todas as mensagens pendentes, setando o *bit* ABAT do registrador CANCTRL. Para continuar a transmitir as mensagens, o usuário deve zerar o *bit* ABAT.

Apenas mensagens que ainda não começaram a transmissão podem ser abortadas. Uma vez iniciada a transmissão, não será possível para o usuário resetar o *bit* TXREQ.

8.3 Recepção de Mensagens

O MCP2510 possui dois *buffers* de recepção, o RXB0 e o RXB1. Cada um desses *buffers* possuem múltiplos filtros de aceitação associados. Existe também o MAB (*Message Assembly Buffer*), que age como um terceiro *buffer*. Ele está sempre pronto para receber uma mensagem do barramento. Todas as mensagens recebidas são agrupadas e após serem aceitas por um ou mais filtros são transferidas para um dos outros *buffers* de recepção.

Quando uma mensagem é movida para um dos *buffers* de recepção, uma interrupção será gerada no pino /INT caso essa interrupção esteja habilitada. Quando o MCU concluir o processamento dessa mensagem, o *flag* de sinalização do respectivo *buffer* deverá ser zerado, afim de permitir que uma nova mensagem seja recebida nesse *buffer*.

8.3.1 Prioridade de Recepção

O RXB0 é o *buffer* que possui maior prioridade de recepção. Existem dois filtros aceitação (RXF0 e o RXF1) associados a ele. O RXB1 tem menor prioridade, e quatro filtros de aceitação (RXF2, RXF3, RXF4 e RXF5) estão associados a ele. Cada um desses *buffers* possui uma máscara de aceitação (RXM0 e o RXM1).

O *buffer* de recepção RXB0 pode ser configurado de tal modo que, caso ele esteja ocupado e outra mensagem válida tenha sido recebida por ele, não ocorra um erro de sobrecarga (*overflow*). Essa nova mensagem será movida para o *buffer* de recepção RXB1, sendo desprezado os valores dos filtros e da máscara do RXB1.

8.3.2 Filtros e Máscaras de Aceitação

As máscaras e filtros são utilizadas para informar se a mensagem carregada no MAB deverá ou não ser transferida para um dos dois *buffers* de recepção. Uma vez no MAB, o ID da mensagem é comparada ao valor do filtro. Em caso positivo, ela é carregada no *buffer* correspondente. A máscara é usada para determinar quais *bits* do filtro serão analisados.

É importante saber que as máscaras e filtros somente podem ser ajustadas quando o MCP2510 está em modo de configuração. Caso ele esteja em modo de funcionamento, os registradores não poderão ser acessados.

8.4 Pinos RX0BF E RX1BF

Os pinos RX0BF e RX1BF podem ser configurados como pinos de interrupção ou como saídas digitais padrão. Quando configurados como pinos de interrupção, eles podem indicar se uma mensagem válida foi carregada nos respectivos *buffers* de recepção RXB0 e RXB1.

8.5 Temporização dos *Bits*

Todos os nós de uma rede CAN devem possuir a mesma taxa de transmissão. Porém, uma mesma taxa de transmissão de dados não significa que necessariamente todos os nós devem ter a mesma freqüência de *clock*. Caso os *clocks* sejam diferentes, a taxa de transmissão de cada nó deve ser ajustada de modo que todas elas sejam iguais. O MCP permite que isso seja feito através do ajuste do BRP (*Baud Rate Prescaler*) e do número de *time quanta* de cada segmento do *bit time*.

O protocolo CAN utiliza a codificação NRZ (*Non-Return to Zero*), que não codifica um *clock* dentro do fluxo de dados. Como os osciladores e conseqüentemente os tempos de transmissão variam de nó para nó, o receptor deve ter algum tipo de PLL (*Phase Locked Loop*) sincronizado com as transições de dados do transmissor para sincronizar e manter o *clock* do receptor. Assim, é necessário o uso do mecanismo de inserção de *bits* para garantir-se que exista ao menos uma transição a cada 6 *bits*. Esse mecanismo garante que o PLL digital (DPLL) não perderá o sincronismo. Observe que o DPLL mantém a sincronização com as transições dos *bits*, e não os nós entre si. Os nós não mantêm sincronia, somente transmitem dados a uma mesma taxa.

A temporização do MCP2510 é implementada usandose um DPLL que é configurado para se sincronizar com os dados que estão sendo recebidos, e para prover uma temporização nominal para que seja feita a transmissão de mensagens. Esse DPLL divide o *bit time* em múltiplos segmentos que duram determinadas unidades de tempo. Essas unidades de tempo são os *time quanta* (TQ).

A maior taxa de transmissão é definida pelo protocolo como sendo de 1 Mbps. Assim, o tempo nominal de um *bit* (*bit time*) é definido por:

$$T_{bit} = 1/Taxa$$

O *bit time* é formado dos seguintes segmentos:

- Segmento de sincronização (Seg_Sinc);
- Segmento de propagação (Seg_Prop);
- Segmento de fase 1 (Seg_Fase1);
- Segmento de fase 2 (Seg_Fase2).

Deste modo, o *bit time* e dado também pela equação:

$$Tbit = T_{Seg\_Sinc} + T_{Seg\_Prop} + T_{Seg\_Fase1} + T_{Seg\_Fase2}$$

O protocolo exige que um *bit time* tenha ao menos 8TQ de duração, e no máximo 25TQ. Ele também define que o tempo nominal mínimo de um *bit* é de 1s, levando a uma taxa nominal máxima de 1Mbps.

O *time quanta* (TQ) é uma unidade fixa de tempo que é derivada do oscilador do dispositivo (MCP2510). A equação a seguir define como calcular o *time quanta*.

$$TQ = 2 * (BRP + 1) * T_{osc}$$

Onde:

BRP é o *Baud Rate Prescaler* e deve ser um valor inteiro entre 1 e 64.

Tosc é o período do oscilador do dispositivo.

A configuração da temporização dos *bits* é feita utilizando os registradores CNF1, CNF2 e CNF3 do MCP2510.

8.6 Detecção de Erro

O MCP2510 implementa todos os sofisticados mecanismos de detecção de erro utilizados pelo protocolo CAN. Pode-se detectar os seguintes tipos de erro:

- Erro de CRC;
- Erro de reconhecimento;
- Erro de formato;
- Erro de *bit*;
- Erro de inserção de *bits*.

Ele contém dois contadores de erro: o TEC (Contador de Erro de Transmissão) e o REC (Contador de Erro de Recepção). De acordo com os valores desses contadores, o nó pode ser classificado como um nó ativo, passivo ou desconectado. Os registradores TEC e REC do MCP registram esses tipos de erro e o registrador EFLG sinaliza os possíveis estados que o controlador CAN de um nó está classificado no momento.

8.7 Interrupções

O MCP2510 possui oito fontes de interrupção. Quando ocorre uma interrupção o pino INT vai para um nível baixo de tensão, por ser ativo baixo, somente se o *bit* de interrupção dessa interrupção estiver habilitado, e permanece nesse estado até que o microcontrolador limpe o seu respectivo *flag* de interrupção. Esse sinalizador não pode ser limpo até que a condição que o gerou seja removida.

É recomendável que se use a instrução de modificação de *bits* (*bit modify*) que será mostrada na seção que aborda a SPI, ao invés de simplesmente escrever no registrador de *flags* de interrupção. Esse procedimento evita que interrupções sejam perdidas por escritas indevidas nos *flags*. O registrador CANINTE contém os *bits* que habilitam cada fonte de interrupção, e o registrador CANINTF contém os *bits* que sinalizam as interrupções que ocorreram.

E importante esclarecer que uma interrupção no pino INT do MCP2510 ocorrerá somente se o respectivo *bit* da interrupção (registrador CANINTE) estiver setado. Caso contrário, apenas haverá uma sinalização do evento que geraria uma interrupção (registrador CANINTF). As interrupções fornecidas pelo MCP2510 são:

- **Interrupção de Transmissão** - Essa interrupção é gerada toda vez que um *buffer* de transmissão for esvaziado, indicando que uma nova mensagem pode ser carregada nesse *buffer* de transmissão. Existe uma interrupção para cada *buffer* de transmissão.
- **Interrupção de Recepção** - Essa interrupção é gerada toda vez que um *buffer* de recepção receber uma mensagem válida. Existe uma interrupção para cada *buffer* de recepção.
- **Interrupção de Erro de Mensagem** - Essa interrupção é gerada toda vez que ocorrer um erro na transmissão ou recepção de uma mensagem.
- **Interrupção de Atividade no Barramento** - Quando o MCP2510 está no modo *sleep* e ocorre alguma atividade no barramento, essa interrupção é gerada.
- **Interrupção de Erro** - Quando ocorre um *overflow* ou uma mudança no estado de erro do controlador CAN, essa interrupção é gerada.

8.8 Modos de Operação

A configuração do modo de operação do MCP2510 é feita utilizando o registrador CANCTRL. O registrador CANSTAT indica qual o modo de operação atual do dispositivo. Existem 5 modos de operação:

- **Modo de Configuração** - Esse é o modo que o MCP2510 se encontra após ser ligado ou após um *reset*. Ao entrar nesse modo todos os registradores de contagem de erro são zerados. Somente nesse modo de operação pode-se configurar os registradores de máscara e filtro de aceitação, de temporização de *bit* e os modos de configuração dos pinos TXnRTS e RXnBF.

- **Modo Normal** - Este é o modo de operação padrão do MCP2510. Neste modo o dispositivo monitora todas as mensagens no barramento CAN e gera *bits* de reconhecimento, mensagens de erro e de sobrecarga.

- **Modo *Sleep*** - Esse modo é utilizado para minimizar o consumo do dispositivo. Nesse modo, a SPI permanece ativa, permitindo acesso aos registradores. Pode-se sair desse modo quando ocorrer atividade no barramento, ou utilizando o microcontrolador. Deve-se ter cuidado para não entrar nesse modo quando o MCP2510 estiver transmitindo uma mensagem. Caso ocorra, a transmissão será interrompida, erros ocorrerão no barramento e a mensagem permanecerá pendente.

- **Modo *Listen-only*** - Nesse modo nenhuma mensagem poderá ser transmitida, inclusive *flags* de erro. Utiliza-se esse modo para monitoração do barramento ou para detecção da taxa de transmissão. Ele recebe todas as mensagens, inclusive as mensagens de erro. Os contadores de erro são zerados e desativados, e os filtros e as máscaras são usados apenas para controlar as mensagens que serão recebidas nos *buffers* de recepção.

- **Modo *Loopback*** - Este modo é utilizado no desenvolvimento do sistema. Nele permite-se que sejam feitas transmissões internas de mensagens do *buffer* de transmissão para o *buffer* de recepção, sem transmitir para o barramento CAN.

8.9 Interface SPI

O MCP2510 foi projetado para se comunicar com outros dispositivos pela *interface* SPI (*Serial Peripheral Interface*). Ele suporta os modos 0,0 e 1,1, e está configurado para operar como dispositivo escravo. Os comandos são recebidos pelo pino SI (*Slave In*), com os *bits*sendo lidos na transição positiva do SCK (*clock*). Os dados são transmitidos pelo pino SO (*Slave Out*) na transição negativa de SCK, e devem ser lidos pelo dispositivo mestre na transição positiva de SCK. O pino CS (*Chip Select*) deve está em nível baixo quando uma transmissão for efetuada. Esse controlador CAN possui 6 instruções SPI: instrução de leitura, escrita, requisição de transmissão, leitura de status, modificação de *bit* e *reset*. Nas subseções seguintes explicaremos cada uma dessas instruções.

8.9.1 Instrução de Leitura

O comando de instrução de leitura, que corresponde ao valor 0x03 em hexadecimal, deverá ser enviado para o MCP2510, e em seguida o endereço do registrador do controlador que se deseja efetuar a leitura dados. Após isso, o controlador retornará pelo pino SO o valor atual desse registrador. Um apontador de endereços interno ao MCP2510 é incrementado logo após a transferência do dado, para o registrador seguinte. Portanto, é possível ler o próximo registrador. Para isso basta continuar os pulsos de *clock* e manter o pino CS em nível baixo. Na figura 8.3 são ilustrados os esquemas com os valores dos pinos da *interface* SPI em uma instrução de leitura.

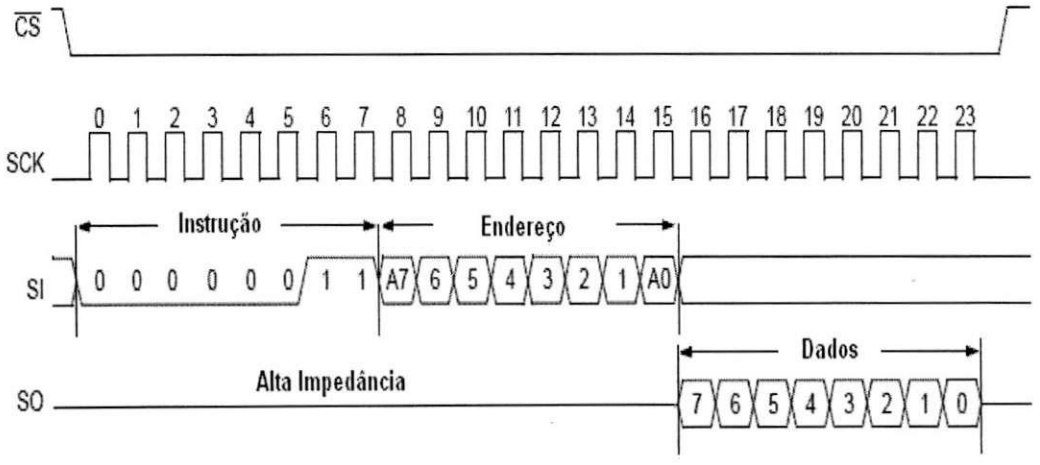

Figura 8.3: Diagrama com os Sinais dos Pinos da *Interface* SPI em uma Instrução de Leitura.

8.9.2 Instrução de Escrita

O comando de instrução de escrita, que corresponde ao valor 0x02 em hexadecimal, deverá ser enviado para o MCP2510, em seguida o endereço do registrador do controlador que se deseja fazer a escrita e por último o valor que se deseja escrever nesse registrador. Toda transmissão de dados é feita pelo pino SI. Enquanto isso, o pino SO ficará em nível de alta impedância. É possível continuar escrevendo nos registradores consecutivos. Para isso, basta manter o *clock*, e o pino CS em nível baixo. Na figura 8.4 são ilustrados os esquemas com os valores dos pinos da *interface* SPI em uma instrução de escrita.

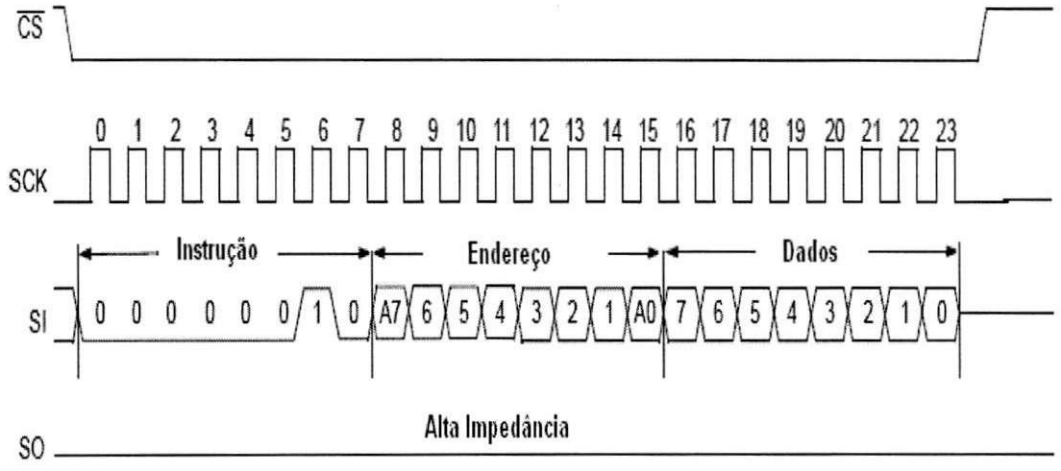

Figura 8.4: Diagrama com os Sinais dos Pinos da *Interface* SPI em uma Instrução de Escrita.

8.9.3 Instrução de Requisição de Transmissão

Essa instrução pode ser usada para iniciar a transmissão de uma mensagem de um ou mais *buffers* de transmissão. Como existem 3 *buffers* de transmissão, essa instrução possui 3 valores diferentes de comandos. Para iniciar a transmissão do *buffer* TXB0 ou TXB1 ou TXB2, deve-se enviar, respectivamente, os valores 0x81, 0x82 ou 0x83. Na figura 8.5 são ilustrados os esquemas com os valores dos pinos da *interface* SPI em uma instrução de requisição de transmissão.

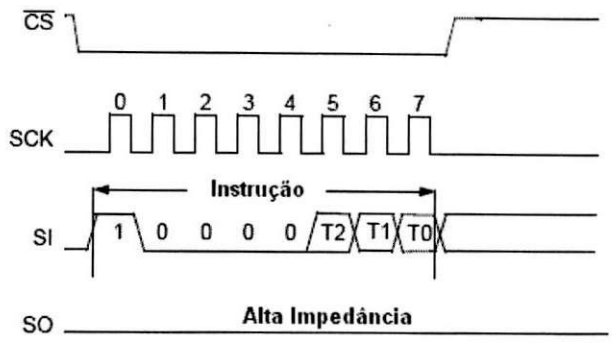

Figura 8.5: Diagrama com os Sinais dos Pinos da *Interface* SPI em uma Instrução de Requisição de Transmissão.

8.9.4 Instrução de Leitura de *Status*

Essa instrução permite que seja feita a leitura de determinados *bits* de registradores usados na transmissão e recepção de mensagens. O comando de instrução de leitura de *status*, que corresponde ao valor 0xA0 em hexadecimal, deverá ser enviado para o MCP2510 pelo pino SI, e em seguida o controlador enviará pelo pino SI o valor dos *bits* desses registradores. Na figura 8.6 são ilustrados os esquemas com os valores dos pinos da *interface* SPI em uma instrução de leitura de *status*.

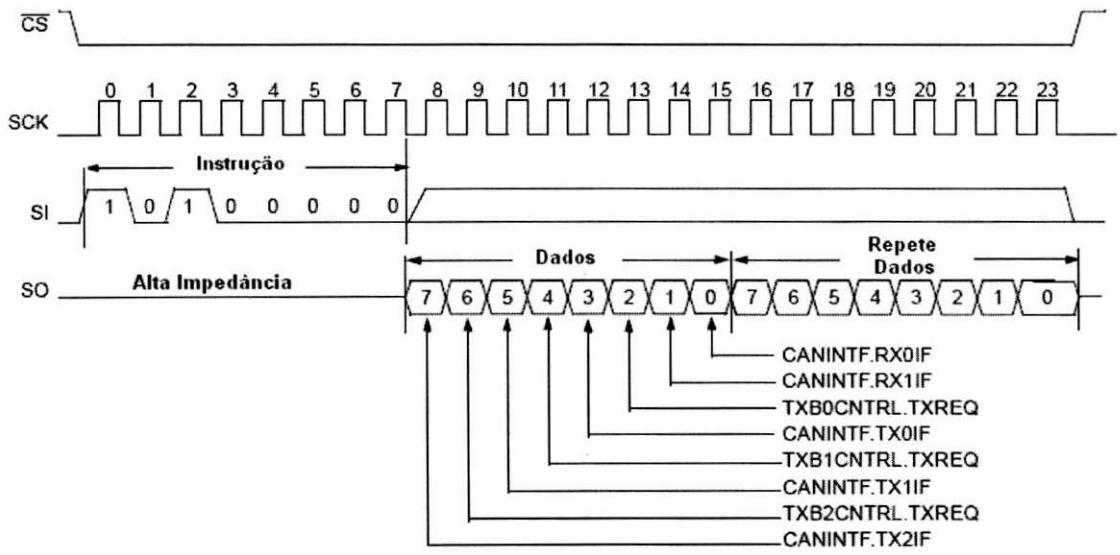

Figura 8.6: Diagrama com os Sinais dos Pinos da *Interface* SPI em uma Instrução de Leitura de *Status*.

8.9.5 Instrução de Modificação de *Bit*

Essa instrução permite que seja feita a modificação de determinados bits de um registrador. Os *bits* que serão modificados serão indicados por uma máscara. Primeiro deve-se enviar o valor do comando dessa instrução, que corresponde a 0x05 em hexadecimal, em seguida o endereço do registrador que se deseja realizar a escrita, o valor da máscara e por último o dado a ser escrito nesse registrador. Esses dados serão transmitidos pelo pino SI e o pino SO permanecerá em estado de alta impedância. Nas figuras 8.7 e 8.8 são ilustrados, respectivamente, o procedimento de funcionamento da máscara e o esquema com os valores dos pinos da SPI em uma instrução de modificação de *bit*.

Figura 8.7: Exemplo de Funcionamento da Máscara em uma Instrução de Modificação de *Bit*.

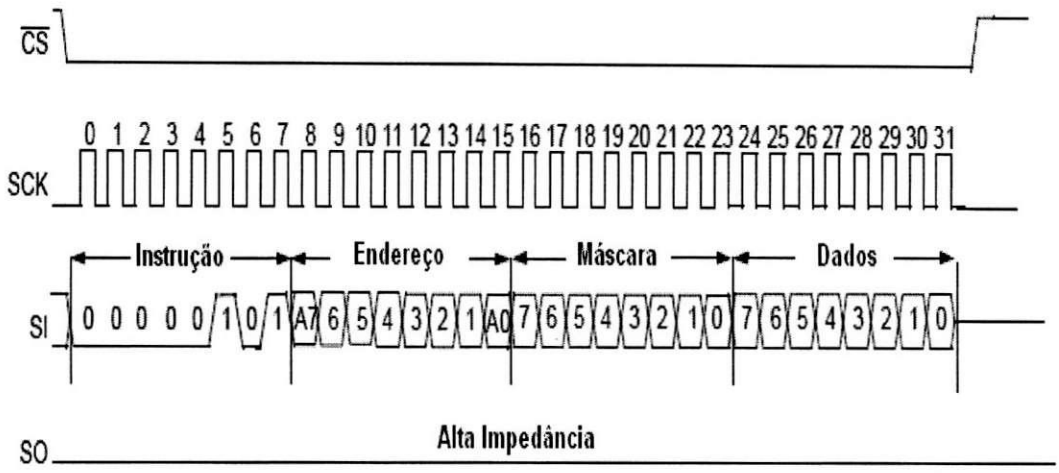

Figura 8.8: Diagrama com os Sinais dos Pinos da *Interface* SPI em uma Instrução de Modificação de *bit*.

8.9.6 Instrução de *Reset*

Essa instrução pode ser usada para re-inicializar os registradores internos e para setar no modo de configuração o MCP2510. Este comando possui a mesma funcionalidade que o pino *RESET*. Após o *reset*, o MCP2510 será mantido em *reset* durante 128 ciclos do seu oscilador. O comando da instrução *reset*, que corresponde a 0xC0 em hexadecimal, deverá ser enviado ao controlador pelo pino SI. Na figura 8.9 são ilustrados os esquemas com os valores dos pinos da SPI em uma instrução de *reset*.

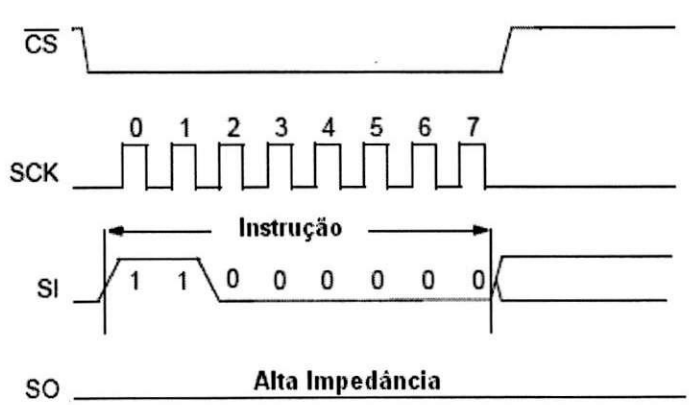

Figura 8.9: Diagrama com os Sinais dos Pinos da *Interface* SPI em uma Instrução de *Reset*.

Capítulo 9

ANEXO D - Transceiver CAN MCP2551

9.1 Introdução

O MCP2551 é um dispositivo de alta velocidade e tolerante a falhas, que atua como *interface* entre o protocolo CAN e o barramento físico. Esse transceiver é compatível com o padrão ISO11898 e pode operar a taxas de até 1 Mbps. Cada nó em um sistema CAN deve ter um dispositivo para converter os sinais digitais gerados por um controlador em sinais que serão transmitidos por um barramento diferencial. É possível ter em um mesmo barramento CAN que utilize esse transceiver, um máximo de 112 nós. Nas figuras 9.1 e 9.2 são ilustrados, respectivamente, a pinagem e um diagrama de bloco do MCP2551.

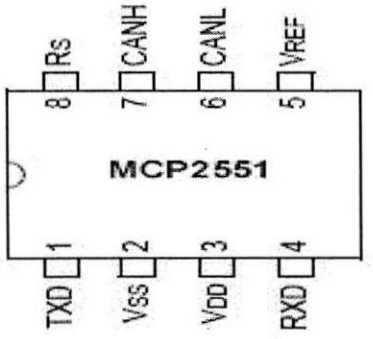

Figura 9.1: Esquema da Pinagem do MCP2551.

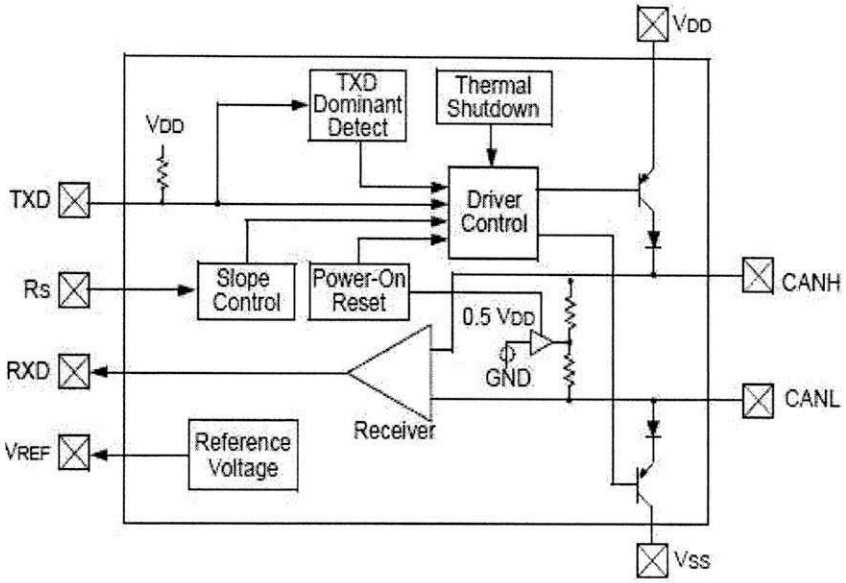

Figura 9.2: Diagrama de Blocos do MCP2551.

9.1.1 Modos de Operação

O pino Rs do MCP2551 permite que o mesmo seja configurado em um dos 3 possíveis modos de operação. Esses modos são:

- ***High-Speed*** - Esse modo é selecionado conectando o pino Rs a terra. Esse modo deve ser utilizado quando as taxas de transmissões no barramento forem altas;
- ***Slope - Control*** - Esse modo é selecionado conectando um resistor externo entre o pino Rs e o terra. Esse modo é utilizado para reduzir as interferências eletromagnéticas;
- ***Standby*** - Esse modo é selecionado aplicando-se um nível alto no pino Rs. Nesse modo o pino Tx é desligado e o pino Rx opera em baixa corrente.