UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

Centro de Engenharia Elétrica e Informática
Departamento de Engenharia Elétrica

Daniel Cardoso de Morais

Controlador CAN em VHDL

Campina Grande
Julho de 2012

Universidade Federal de Campina Grande
Departamento de Engenharia Elétrica

Daniel Cardoso de Morais

Controlador CAN em VHDL

Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Orientador:

Prof. Dr. Marcos Ricardo Alcântara Morais

Campina Grande
Julho de 2012

Universidade Federal de Campina Grande
Departamento de Engenharia Elétrica

Daniel Cardoso de Morais

Controlador CAN em VHDL

Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Data de Aprovação:

24 de Julho de 2012

Banca Examinadora:

Prof. Dr. Alexandre Cunha Oliveira
Universidade Federal de Campina Grande

Prof. Dr. Marcos Ricardo Alcântara Morais
Universidade Federal de Campina Grande

Campina Grande
Julho de 2012

Aos meus pais, Misael e Neuza, e à minha namorada, Be, que sempre esteve ao meu lado nos momentos em que precisei.

Agradecimentos

Agradeço principalmente à minha família, que sempre esteve ao meu lado quando precisei e pelo exemplo que sempre foi para minha vida. Agradeço também, à minha namorada, Be, que esteve sempre ao meu lado, me incentivando a continuar sempre que pensei em desistir.

Ao instituto Fraunhofer IESE, por ter aberto as portas e me recebido para realizar as atividades deste trabalho.

Aos professores do Departamento de Engenharia Elétrica, por terem me dado toda a base profissional que me proporcionou o desenvolvimento deste trabalho e dos futuros que virão. Em especial ao professor Marcos Ricardo Alcântara Morais, por ter me orientado neste trabalho e com quem aprendi bastante.

Aos funcionários do Departamento de Engenharia Elétrica, que sempre nos incentivaram e apoiaram.

Aos meus colegas de curso, que me acompanharam ao longo de cinco anos, me ajudando a chegar no final em longas noite sem dormir.

Lista de Siglas

ACK: Reconhecimento, em inglês *Acknowledgment* – parte do frame utilizado para confirmar o seu recebimento.

BRP: Taxa de Divisão, em inglês *Baud Rate Prescaler* - define o comprimento do *time quantum*.

CAN: *Controller Area Network* – protocolo de comunicação desenvolvido para ser utilizado em barramentos de veículos.

CRC: Checagem de Redundância Cíclica, em inglês *Cyclic Redundant Check* – teste de verificação de redundância cíclica, utilizado para verificar se o frame transmitido é o mesmo do recebido.

CSMA/DCR: Acesso múltiplo com sensoriamento da portadora com resolução de colisão determinística, em inglês *Carrier Sense Multi-Access with Deterministic Collision Resolution* - protocolo de comunicação que tenta evitar número de colisões, nele as mensagens só são transmitidas quando o barramento está livre.

DLC: Código do Tamanho do Dado, em inglês *Data Length Code* – indica o tamanho da palavra de dados transmitida.

ECU: Unidades de Controle Eletrônico, em inglês *Electronic Control Units* – dispositivo eletrônico utilizado no controle de dispositivos mecânicos e eletrônicos de um automóvel.

EOF: Fim da Mensagem, em inglês *End Of Frame* – conjunto de sete bits que indicam o fim da mensagem.

FIFO: Primeiro a Entrar, Primeiro a Sair, em inglês *First in, First out* - estrutura de dados tipo fila, onde o primeiro a entrar é o primeiro sair.

FPGA: Arranjo de Portas Programável em Campo, em inglês *Field Programmable Gate Array*: circuito integrado projetado para ser configurado por um usuário após a fabricação.

IDE: Extensão do Identificador, em inglês *Identifier Extension* – identifica quando uma mensagem é enviada no formato estendido.

IOB: Blocos de entrada e saída, em inglês *Input/Output Block* – são buffers que funcionarão

como um pino bidirecional de entrada e saída do FPGA.

IP-core: Módulo de Propriedade Intelectual, em inglês *Intellectual Property Core* – bloco reutilizável de uma unidade lógica, célula ou projeto de layout de chip.

LUT: Tabela de Busca, em inglês *Look-Up Table*- tipo de bloco lógico que contém células de armazenamento que são utilizadas para implementar pequenas funções lógicas.

MUX: Multiplexador – dispositivo que codifica as informações de duas ou mais fontes de dados num único canal.

NRZ: Não Retorna a Zero, em inglês *No Return to Zero* – tipo de codificação que representa os símbolos digitais (0 e 1) como dois níveis de tensão diferente de zero.

RJW: Largura do Salto de Ressincronização, em inglês *Resynchronization Jump Width* - deslocamento máximo que o controlador pode usar para fazer a sincronização com o barramento.

RTR: Solicitação Remota de Transmissão, em inglês *Remote Transmit Request* - bit utilizado para solicitar uma mensagem a um nó.

SAE: Sociedade de Engenheiros de Automotivos, em inglês *Society of Automobile Engineers* - associação mundial de engenheiros e especialistas técnicos relacionados aos setores aeroespacial, automotivo e de veículos comerciais.

SOC: Sistema em um Chip, em inglês *System-on-Chip* - sistema eletrônico que realiza funções digitais, analógicas, mistas ou de radiofrequência, contido em um circuito integrado.

SOF: Início de Mensagem, em inglês *Start of Frame* - bit utilizado para indicar o início da transmissão.

Glossário

ACK Slot - abertura para o reconhecimento, bit responsável pela indicação de que a mensagem foi reconhecida e recebida.

Bit - unidade mínima de informação em um sistema digital, geralmente 0 e 1.

Broadcast - difusão, quando as mensagens são transmitidas para mais de uma estação de comunicação ao mesmo tempo.

Bus Idle - barramento livre, indica quando nenhuma mensagem está sendo transmitida pelo barramento.

Bus off - barramento desligado, indica que o controlador foi desconectado do barramento por apresentar uma grande quantidade de erros.

Bytes - sequência formada por um número fixo de bits, em geral oito.

Clock - relógio, tempo usado como base para o funcionamento de um dispositivo digital.

Flag - bandeira, nomeação utilizada para sinais de erro ou de controle.

Kit - conjunto de equipamentos para um uso específico reunidos em uma embalagem adequada.

Time quantum - quantum de tempo, menor quantidade de tempo utilizado na geração de uma mensagem.

Resumo

A rede CAN (*Controller Area Network*) foi desenvolvida por Robert Bosch GmbH, em meados da década de 1980, com o propósito de ser utilizada em comunicação de ECU (*Electronic Control Units*) em veículos. Hoje é amplamente utilizada em várias áreas. A proposta deste trabalho é desenvolver um controlador de rede CAN utilizando a linguagem VHDL. O projeto do controlador é capaz de construir mensagem, arbitrar, verificar erros e enviar mensagens de erro. A verificação funcional é feita utilizando o Xilinx ISim 12.3, que simula o controlador se comunicando com outro. Por fim, o controlador é implementado na placa XUPV5-LX110T da Xilinx para realizar o teste de compatibilidade ao se comunicar com um outro dispositivo CAN, o USB-to-CAN compact da IXXAT.

Palavras-chave: Rede CAN, IP-core, FPGA, Controlador CAN.

Abstract

The CAN (Controller Area Network) network was developed by Robert Bosch GmbH, in mid-1980, with the purpose of be used in communication of ECUs (Electronic Control Units) to vehicles. Today it is widely used in various areas. This work aims to develop a CAN network controller using VHDL. The controller project is able to build message, arbitration, error checking and sending error messages. The functional verification is performed using the Xilinx ISIM 3.12 simulating the controller communicating with each other. Finally, the controller is implemented on-board XUPV5 LX110T of Xilinx, for testing compatibility when communicating with another CAN device, the USB-to-CAN compact of IXXAT.

Keywords: CAN network, IP-core, FPGA, CAN controller.

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

Nas últimas cinco décadas, o número e a sofisticação dos sensores e componentes eletrônicos nos veículos têm aumentado consideravelmente. Analistas estimam que mais de 80% da automação é devida à eletrônica (HEFFERNAN & LEEN 2002). Um bom exemplo é o número de Unidades de Controle Eletrônico (ECU) em carros da Mercedes, BMW, Audi e Volkswagen, que aumentaram de cinco ou menos no começo da década de 1900 para cerca de 40 na virada do milênio (DAVIS ALAN BURNS & LUKKIEN 2007).

Uma das grandes causas desse crescimento foi o impulso que a indústria de eletrônicos sofreu nesses anos. O mercado solicitava mais componentes em um espaço menor de tempo. Um novo método de desenvolvimento chamado System-on-Chip (SOC), torna possível que todos os sistemas sejam integrados e implementados em um único chip (SANTOS 2009). Para aplicar essa metodologia, são utilizados componentes previamente projetados e verificados, denominados Módulo de Propriedade Intelectual (IP-core). Esse módulo torna possível a entrega dos projetos no tempo esperado e determinado pelo mercado.

A facilidade de criar novos componentes possibilitou que os veículos possuíssem mais ECU, consequentemente, uma maior quantidade de redes de comunicação integrados para cada sistema. Assim, no começo da década de 1980, Robert Bosch GmbH desenvolveu uma rede baseada em comunicação serial denominada *Controller Area Network* (CAN) para redes internas de veículos. Essa rede deveria conectar todos os ECU no mesmo barramento, reduzindo o número de fios necessários no carro.

A rede CAN hoje é amplamente utilizada em varias áreas, não se limitando apenas a redes internas de veículos. Por exemplo, o protocolo CAN vem sendo usado na indústria (FREDRIKSSON 2012), em robôs (BOURDON & DELAPLACE 1996), em residências (MORAES & JUNIOR 2001), em transportes, em equipamentos médicos, na agricultura, entre outros (CIA 2011). Em geral, a rede CAN é utilizada quando vários subsistemas com capacidade de decisão precisam de um mecanismo de comunicação.

Devido à sua vasta gama de utilização, um controlador CAN pode ser usado em testes de

desenvolvimento de novas ferramentas. Por exemplo, ao desenvolver um IP-core para aplicações industriais, haverá um momento em que serão realizados testes com outros dispositivos para validar o seu funcionamento. A criação de um novo controlador a cada projeto desenvolvido pode interferir gravemente no seu cronograma. A proposta deste trabalho é desenvolver um controlador CAN capaz de enviar e receber as mensagens para a realização desses testes.

No Capítulo 2 são apresentados os objetivos deste trabalho. No Capítulo 3, é feita a revisão teórica do assunto necessária para o entendimento do protocolo CAN. A apresentação das soluções encontradas está descrita no Capítulo 4. Por fim, no Capítulo 5, serão apresentadas a lógica encontrada para o funcionamento do controlador e os resultados obtidos nos testes e simulações.

2 Objetivos

2.1 Objetivo geral

Criar um controlador de rede CAN que possa ser reutilizável. Para isso, será desenvolvido um IP-core utilizando a linguagem VHDL para descrever o controlador.

2.2 Objetivos específicos

Ao desenvolver um IP-core que necessite de uma comunicação com o barramento CAN, a utilização de um produto encontrado no mercado pode ser inviável, pois a grande maioria deles é feita para a utilização em computadores. Portanto, o controlador deve ter características genéricas, para que seja compatível com diversos sistemas.

É objetivo desse trabalho é desenvolver um controlador que atenda aos seguintes requisitos funcionais:

- Ser capaz de codificar e decodificar a mensagem no formato padrão,
- Requisitar uma mensagem,
- Criar mensagens de erro,
- Caso detecte um grande número de erros, deve ser desconectado do barramento.

3 Referencial teórico

Neste capítulo, são introduzidos os conceitos gerais para entender o funcionamento do protocolo CAN. Na seção 3.1, é apresentado o contexto histórico de surgimento da rede CAN. Na seção 3.2, introduz-se o seu funcionamento básico. Por fim, na seção 3.3, mostra-se de forma mais detalhada as especificações do protocolo.

3.1 Contexto histórico

A facilidade de criação de novos componentes possibilitou o surgimento de mais ECU, consequentemente o número de redes integradas para cada sistema também cresceu. Logo, o número de fios para a comunicação também aumentou. Cada 50 quilogramas extras de fio eleva o consumo de combustível em 0,2 litro por 100 quilômetros rodados (HEFFERNAN & LEEN 2002).

No começo da década de 1980, os engenheiros da Bosch começaram a desenvolver um novo protocolo de barramento para rede em veículos. Em 1986, o CAN foi apresentado no congresso Sociedade de Engenheiros Automotivos (SAE) em Detroit. Um ano depois, Intel e Philips Semiconductor introduziram no mercado os primeiros controladores CAN em chips: o 82526 e o 82C200, respectivamente. Em 2004, existiam pelo menos 15 fabricantes, com um total de mais de 50 tipos diferentes de famílias de microprocessadores com protocolo CAN implementados (DAVIS ALAN BURNS & LUKKIEN 2007).

A Bosch submeteu para padronização internacional a segunda versão da especificação CAN no começo da década de 1990 (CIA 2011). Com a ISO 11898 publicada em 1993, o protocolo CAN foi definido fisicamente para atingir taxas de até 1 Mbit/s; e o padrão foi estendido com a adição de 29 bits no identificador da mensagem.

Os carros da série S de 1991 da Mercedes foram os primeiros a utilizar a rede CAN (CIA 2011). O protocolo CAN foi adotado por outras fábricas, incluindo Volvo, BMW, Audi, Saab, Ford, Volkswagen, Renault e Fiat. Hoje, quase todos os carros fabricados na Europa

são equipados com pelo menos um barramento CAN (HEFFERNAN & LEEN 2002).

O CAN foi desenvolvido para uma comunicação simples, eficiente e robusta. Porém, não foi apenas em redes de veículos que o protocolo foi utilizado. Outros setores da indústria começam a usar esse protocolo em suas redes: a indústria têxtil foi pioneira no uso do CAN (FREDRIKSSON 2012), utilizando a rede na comunicação interna de máquinas. Atualmente, a rede é usada na agricultura, saúde, ciência, construção e em outros setores.

3.2 A rede CAN

A rede CAN é baseada em uma comunicação difusão (*broadcast*), isto significa que todas as mensagens são transmitidas para todos os nós, ou estações de comunicação, ao mesmo tempo. O nó receberá a mensagem e decidirá se irá usá-la ou descartá-la, como mostra a Figura 3.1.

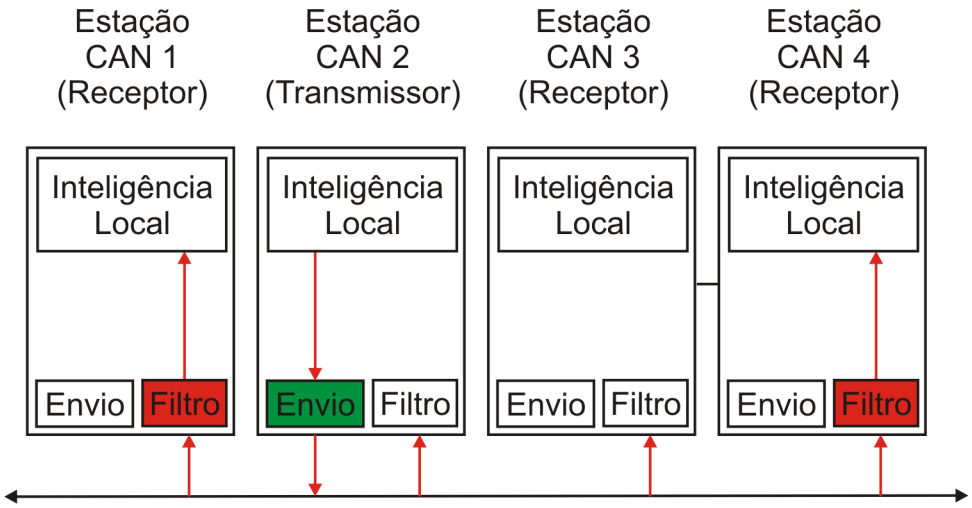

Figura 3.1 *– Funcionamento da comunicação difusão (*broadcast*) na rede CAN, baseado em CIA 2011.*

CAN é uma rede *Carrier Sense Multi-Access with Deterministic Collision Resolution* (CSMA/ DRC), isto é, os nós atrasam a transmissão se o barramento estiver ocupado, mas, assim que a condição de barramento livre for detectada, qualquer nó pode reiniciar a transmissão (SA & NETO 2005). Outro conceito importante é a utilização do não-retorna-a-zero (NRZ), o barramento não retorna para zero para enviar o próximo bit. A representação dos estados lógicos é feita usando duas tensões positivas diferentes, assim, quando o barramento está em zero significa que não está operando corretamente, e a mensagem não é enviada.

A Bosch especificou dois estados - recessivo e dominante - para representar os níveis lógicos. Estes são especificados em International Standards Organisation 1993, para serem representados pela diferença de tensão. A implementação é feita utilizando dois fios, chamados de

CANL e CANH. O bit recessivo é definido quando os dois fios estão no mesmo nível de tensão. O dominante, quando os fios estão em níveis de tensão diferentes.

A prioridade, que compara a mensagem transmitida com uma outra mensagem de menos urgência, é especificada pelo identificador de cada mensagem (CIA 2011), sendo de 11 bits no formato padrão ou 29 bits no formato estendido. O identificador que possuir o menor número binário terá a maior prioridade, assim como o formato padrão possui prioridade com relação ao estendido. Este trabalho implementará apenas o formato padrão.

Para entender melhor, veja o exemplo a seguir, suponha que três nós desejam transmitir uma mensagem ao mesmo tempo (Figura 3.2). No quinto bit do identificador, o primeiro nó escreve um bit recessivo, enquanto os demais nós, um bit dominante. Neste caso, o primeiro nó perde a arbitragem. Um envio de mensagem tem maior prioridade que uma solicitação de envio, assim, o terceiro nó perde a arbitragem no campo Solicitação Remota de Transmissão (RTR) e o segundo nó ganha a arbitragem.

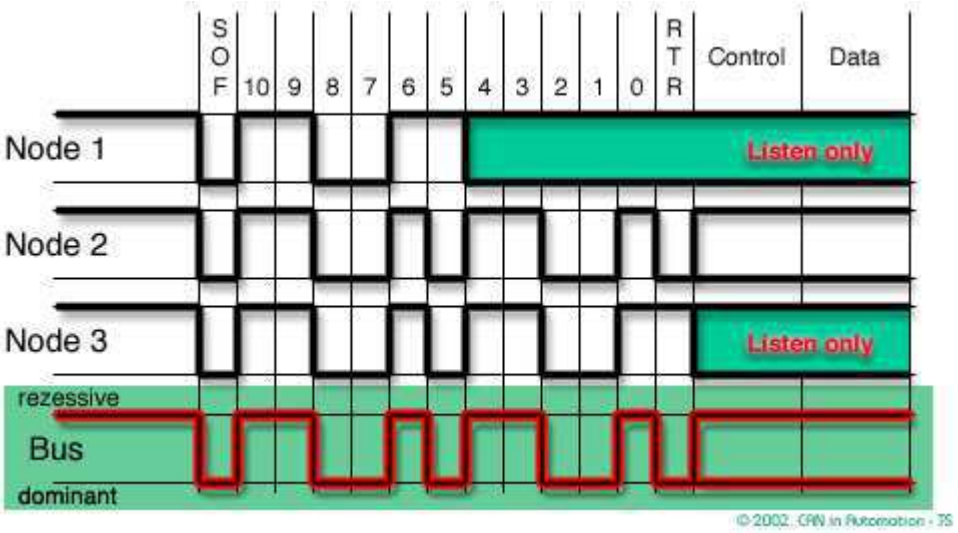

Figura 3.2 *– Arbitragem do barramento (CIA 2011).*

Os erros estão presentes em todos os meios de comunicação, eles podem ocorrer tanto por eventos estáticos quanto por efeitos transientes. De qualquer que seja o tipo, eles precisam ser detectados e corrigidos. O CAN especifica três mecanismos de identificação de erro em nível de mensagem: Checagem de Redundância Cíclica (CRC), Bit de Reconhecimento (ACK bit) e Verificação do Formato das Mensagens. Existem, também, dois mecanismos de detecção de erro em nível de bit: verificação do bit escrito e o bit de enchimento. Se algum erro for detectado, uma mensagem de erro é enviada para alertar os nós sobre possíveis problemas no barramento. Quando a informação continua disponível, a retransmissão é o meio mais eficaz de correção (CAMPBELL 1987).

Cada vez que uma mensagem de erro é enviada, um contador é incrementado por um ou oito, dependendo de como o erro ocorreu (Robert Bosch GmbH 1991). Existem dois contadores a serem incrementados, um para os erros de transmissão e outro para os erros de recepção. O controlador é considerado "*bus off*" quando o contador de transmissão é igual ou maior a 256, não podendo enviar ou receber nenhuma mensagem, prevenindo que um nó problemático perturbe o barramento. Depois de monitorar 11 bits consecutivos recessivos por 128 vezes, o controlador não é mais considerado "*bus off*".

3.3 Especificação CAN

3.3.1 Tipos de Mensagem

As mensagens na rede CAN são divididas em quatro tipos: Mensagem de Dados, Mensagem Remota, Mensagem de Erro e Mensagem de Sobrecarga. A Mensagem de Dados e a Mensagem Remota são separadas por uma mensagem intermediária chamada de Espaço Entre Mensagens (Robert Bosch GmbH 1991).

Mensagem de Dados

A Mensagem de Dados é responsável por enviar os dados para os outros nós. É composta por dez diferentes campos de bits, como mostrado na Figura 3.3.

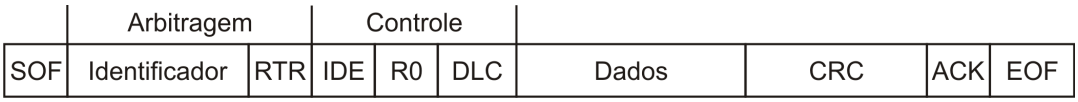

***Figura 3.3** – Campos da Mensagem de Dados*

O Início da Mensagem (SOF) é o primeiro bit a ser enviado e deve ser dominante. Ele indica para a rede que uma mensagem começará a ser enviada.

O segundo campo é o da Arbitragem, que é composto pelo Identificador e RTR, indicando qual mensagem possui maior prioridade no barramento. O Identificador é composto por onze bits, onde os sete mais significantes não podem ser todos recessivos. O bit RTR indica que tipo de mensagem é enviada: como a Mensagem de Dados possui uma maior prioridade, envia esse bit como dominante, e a Mensagem Remota enviará como recessivo.

Como a rede CAN possui dois tipos de extensão para Identificador, a Extensão do Identificador (IDE) é transmitida como dominante para o protocolo-padrão (KOOPMAN 2002). O R0

é um bit reservado e deve ser enviado como recessivo, mesmo que o nó receptor possa recebê-lo como dominante ou recessivo. O campo de Controle termina com o Código do Tamanho do Dado (DLC) e ele é responsável por indicar o número de bytes que o campo Dados deve conter, veja a Tabela 3.1.

***Tabela 3.1** – Número de Bytes do Campo Dados.*

	Código do Tamanho do Dado (DLC)			
Nº de bytes	DLC[3]	DLC[2]	DLC[1]	DLC[0]
0	dominante	dominante	dominante	dominante
1	dominante	dominante	dominante	recessivo
2	dominante	dominante	recessivo	dominante
3	dominante	dominante	recessivo	recessivo
4	dominante	recessivo	dominante	dominante
5	dominante	recessivo	dominante	recessivo
6	dominante	recessivo	recessivo	dominante
7	dominante	recessivo	recessivo	recessivo
8	recessivo	dominante	dominante	dominante

O campo Dados contém os dados que devem ser transmitidos na mensagem. Ele pode conter de 0 a 8 bytes, onde o bit mais significante é mandado primeiro.

Para verificar a mensagem enviada, é feito o CRC de 15 bits e o resultado é transmitido. Os campos SOF, Arbitragem, Controle e Dados são usados para criar o CRC. Para mais informações, veja a sessão 3.3.3. Depois da sequência CRC, é enviado o Delimitador do CRC como recessivo.

O campo Reconhecimento (ACK) é composto por dois bits, o ACK Slot e o Delimitador do ACK. Todas as estações que tenham recebido a sequência CRC correta devem responder subscrevendo o bit recessivo, enviado pelo transmissor, por um bit dominante no ACK Slot. Para terminar o campo ACK, o ACK Delimitador é enviado como recessivo.

O último campo é o Fim da Mensagem (EOF), composto de sete bits recessivos consecutivos.

Mensagem Remota

A Mensagem Remota é utilizada quando um nó deseja que outro lhe envie uma mensagem. Ela é composta pelos mesmos campos da Mensagem de Dados, com exceção do campo de Dados e do RTR, que é enviado como recessivo. A Figura 3.4 mostra os campos presentes na Mensagem Remota.

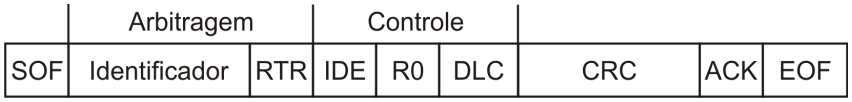

Figura 3.4 *– Campos da Mensagem Remota*

Mensagem de Erro

Uma Mensagem de Erro é enviada quando um erro ocorre na transmissão ou na recepção de uma mensagem. Como mostrado na Figura 3.5, é composta por dois campos: *Flag* de Erro e Delimitador do Erro.

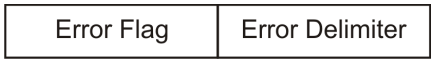

Figura 3.5 *– Campos da Mensagem de Erro*

O *Flag* de Erro é composto de seis bits. Se o controlador for um "erro ativo", um *Flag* de Erro Ativo de seis bits dominantes é enviado. Se o controlador for um "erro passivo", um *Flag* de Erro Passivo de seis bits recessivos é enviado. O Delimitador do Erro consiste em oito bits recessivos.

Mensagem de Sobrecarga

Uma Mensagem de Sobrecarga é gerada por um nó se, devido a condições internas, o nó ainda não é capaz de iniciar a recepção da próxima mensagem ou se durante o intervalo entre as mensagens, um dos dois primeiros bits é dominante (SOFTING 2011). A Figura 3.6 mostra os campos da Mensagem de Sobrecarga.

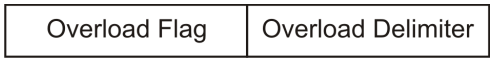

Figura 3.6 *– Campos da Mensagem de Sobrecarga*

O *Flag* de Sobrecarga é composto por seis bits dominantes, e o Delimitador da Sobrecarga consiste em oito bits recessivos, assim como o Delimitador do Erro.

Espaço Entre Mensagens

A Mensagem de Dados e a Mensagem Remota são separadas por uma mensagem chamada Espaço Entre Mensagens. A Mensagem de Erro e a Mensagem de Sobrecarga não necessitam ser separadas por essa mensagem para iniciar a próxima transmissão. Os campos do bit são mostrados na Figura 3.7.

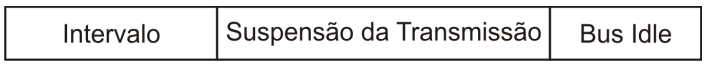

***Figura 3.7** – Campos do Espaço Entre Mensagens*

O Intervalo consiste em três bits recessivos. Durante o Espaço Entre Mensagens, somente é permitido a transmissão de Mensagem de Sobrecarga, as demais mensagens não são permitidas.

Se um nó é "erro passivo" e possui uma mensagem para ser transmitida, é necessário que sejam enviados oito bits recessivos após o Intervalo para poder se tornar *bus idle*. Mas, se uma mensagem iniciar, o nó se tornará receptor da mensagem. Quando o barramento está livre (*bus idle*), o nó pode transmitir a mensagem que estava pendente ou receber uma.

3.3.2 Detecção de Erro

Para a validação das mensagens transmitidas e recebidas, o protocolo CAN especifica cinco tipos possíveis de erro: Erro de Reconhecimento, bit, Erro de Enchimento, CRC e de forma.

- **Erro de Reconhecimento**: ocorre quando um bit dominante não é detectado no ACK Slot.
- **Erro de Bit**: sempre que o transmissor escreve no barramento, ele checa se o barramento possui o mesmo bit escrito. Se o bit não for o mesmo, o erro é indicado. Entretanto, isso não ocorre na arbitragem, quando um bit recessivo é enviado; no ACK Slot; e quando é detectado um bit dominante durante o *Flag* de Erro
- **Erro de Enchimento**: quando uma sequência de cinco bits dominantes ocorre, o próximo bit deve ser recessivo. Da mesma forma, após cinco bits recessivos, o sexto será dominante. Se o nó receptor detectar uma sequência de seis bits consecutivos do mesmo valor, recessivos ou dominantes, um Erro de Enchimento é indicado. O *Flag* de Erro Passivo é detectado como um Erro de Enchimento pelos outros nós.
- **Erro de CRC**: o nó receptor calcula a sequência CRC da mesma forma que o transmissor. Se o calculo da sequência não coincidir com o recebido, um erro de CRC é detectado.
- **Erro de Forma**: é detectado quando um bit ilegal é recebido no CRC delimitador, ACK delimitador e EOF.

3.3.3 Checagem de Redundância Cíclica (CRC)

O CRC é normalmente utilizado para a detecção de erros em redes embarcadas e outros sistemas (KOOPMAN 2002). Ele descreve o byte como um polinômio e então o divide por um polinômio gerador. Na rede CAN, é utilizado o polinômio descrito na Equação 3.1. O resultado é chamado de *sequência CRC*.

$$x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1 \tag{3.1}$$

Na rede CAN, a sequência CRC é gerada no nó transmissor pelos campos SOF, Arbitragem, Controle e Dados. No nó receptor, são utilizados os campos SOF, Arbitragem, Controle, Dados e a sequência CRC para criar uma nova sequência.O resultado é comparado com zero, se for diferente indica que existe ao menos um erro.

3.3.4 Tempo de Bit

A rede CAN utiliza uma comunicação assíncrona, entretanto os nós devem possuir o mesmo tempo de bit. Esse tempo é dividido em quatro partes, representadas na Figura 3.8.

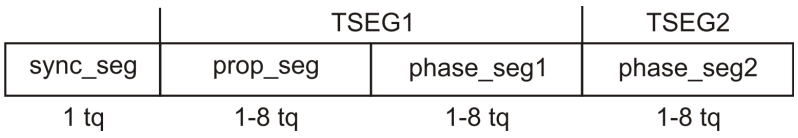

***Figura 3.8** – Tempo do bit*

O **sync_seg** é o tempo para a sincronização de todos os nós do barramento. O **prop_seg** é a parte do tempo para esperar que o sinal seja propagado. O **phase_seg1** e o **phase_seg2** são utilizados para compensar o erro de fase. O somatório do **prop_seg** com **phase_seg1** é chamado de TSEG1; e o **phase_seg2** de TSEG2 (HARTWICH & BASSEMI 1999). O tamanho de cada tempo é mostrado na Figura 3.8, e o total de *time quantum* deve ser entre 8 e 25.

O ponto de amostragem acontece no final do TSEG1. Cada controlador CAN possui um oscilador individual, isso significa que o ponto de amostragem pode acontecer em diferentes instantes de tempo em cada nó. Por causa disso, o deslocamento de fase é diferente em cada nó (SOFTING 2011) para ser possível a sincronização.

3.3.5 Sincronização

Existem duas formas de sincronização na rede CAN: forçada e ressincronização. Como os controladores trabalham com *clocks* diferentes, esses mecanismos são importantes para fazer com que todos leiam o barramento ao mesmo tempo. A borda será usada para a sincronização apenas se o valor detectado diferir do valor do barramento, sendo permitida uma sincronização por tempo de bit.

Quando o barramento está ocioso e é detectada uma mudança de recessivo para dominante acontece a sincronização forçada. Para isso, o tempo do bit é reiniciado para **sync_seg**.

Durante a transmissão de uma mensagem, apenas a ressincronização é permitida. Para realizar a sincronização, ele modifica o tamanho do **phase_seg1** e do **phase_seg2**. Quando a borda ocorre antes do ponto de amostragem, o **phase_seg1** é aumentado. Se a borda ocorre depois do ponto de amostragem, o **phase_seg2** é encurtado. O máximo de deslocamento para o **phase_seg1** e o **phase_seg2** é determinado pela Largura do Salto de Ressincronização (RJW) e deve ser programado entre 1 e 4 *time quantum*.

4 Abordagem de solução

Neste capítulo, abordaremos as soluções propostas para atingir os objetivos do projeto. Este foi desenvolvido em quatro etapas que tratam da proposta da arquitetura, da modelagem do sistema em uma linguagem de descrição de hardware, da simulação e da implementação em FPGA. Os hardwares e softwares necessários para o desenvolvimento do projeto estão descritos na seção 4.1.

O primeiro passo para a criação do controlador foi o desenvolvimento de uma arquitetura para o controlador. O controlador necessita que os dados sejam armazenados antes da transmissão, caso necessite ser reenviado, e após o recebimento de uma mensagem. Ele também deve gerenciar o controle de erro das mensagens recebidas. A descrição dessa arquitetura está na seção 4.2.

O VHDL (VHSIC *Hardware Descript Language*) é uma linguagem de descrição de hardware usada para modelar sistemas eletrônicos. Desenvolvido pelo Departamento de Defesa dos Estados Unidos, VHDL era uma alternativa para substituir os complexos manuais que descreviam o funcionamento dos circuitos integrados construídos para executar uma tarefa específica (WIKIPEDIA 2012). Hoje, a linguagem é padronizada pelo IEEE 1076 - 2008 (IEEE 2009). Assim, a partir da arquitetura proposta, o controlador CAN foi descrito utilizando essa linguagem.

Com a descrição do controlador em linguagem de descrição de hardware, foi realizada a simulação seguindo os passos descritos na seção 4.3 para verificar a funcionalidade. Por fim, o controlador foi implementado conforme a descrição da seção 4.4.

4.1 Recursos utilizados

Atualmente existem diversas empresas fabricantes e especialistas em FPGA, e sua grande maioria também desenvolve softwares compatíveis com suas tecnologias. Dentre esses softwares, o Xilinx ISE 12.3 desenvolvido pela Xilinx será utilizado neste trabalho para a realização da

síntese do RTL. O *kit* de desenvolvimento escolhido foi o XUPV5-LX110T da Xilinx, mostrado na Figura 4.1. Entretanto, esse *kit* cria apenas sinais lógicos, sendo necessário o PCA82C250 para realizar a interface entre o controlador e o barramento físico (SEMICONDUCTORS 2000).

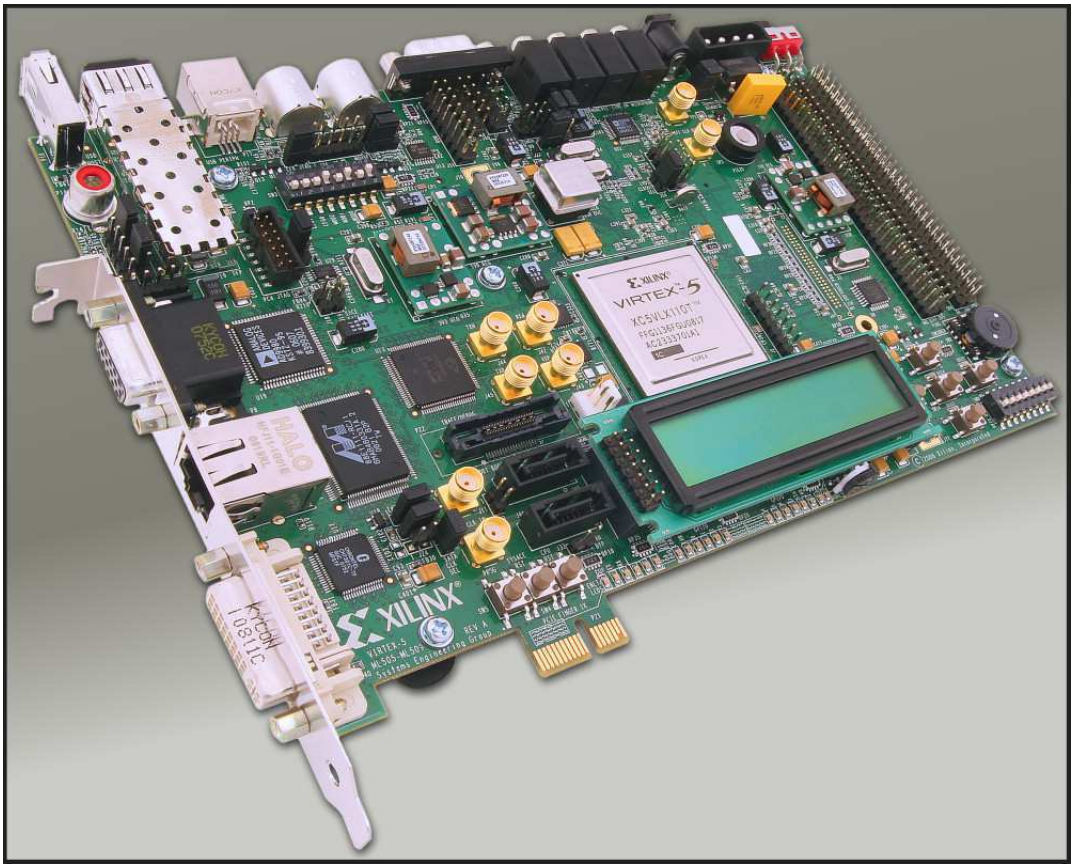

Figura 4.1 *– Kit de desenvolvimento XUPV5-LX110T da Xilinx (XILINX 2011).*

Para comprovar seu funcionamento, o controlador deve se comunicar com o dispositivo USB-to-CAN compact da IXXAT (IXXAT 2011). Essa empresa também fornece o CAN Mini-Monitor V3, que é utilizado para envio e recebimento de mensagens dos seus dispositivos.

4.2 Arquitetura proposta

Partindo da ideia proposta por HARTWICH & BASSEMI 1999, o controlador CAN é dividido em cinco módulos: Divisor de *Clock*, Deserializador, Serializador, Lógica do Tempo de Bit e Processador de Fluxo de Bits. A Figura 4.2 mostra todos os módulos e conexões do controlador.

O *time quantum* (clk_{tq}) utilizado no tempo do bit é gerado pelo Divisor de *Clock*. Tendo como base o *clock* do controlador (clk), este módulo divide-o, seguindo a Equação 4.1, para gerar o clk_{tq}.

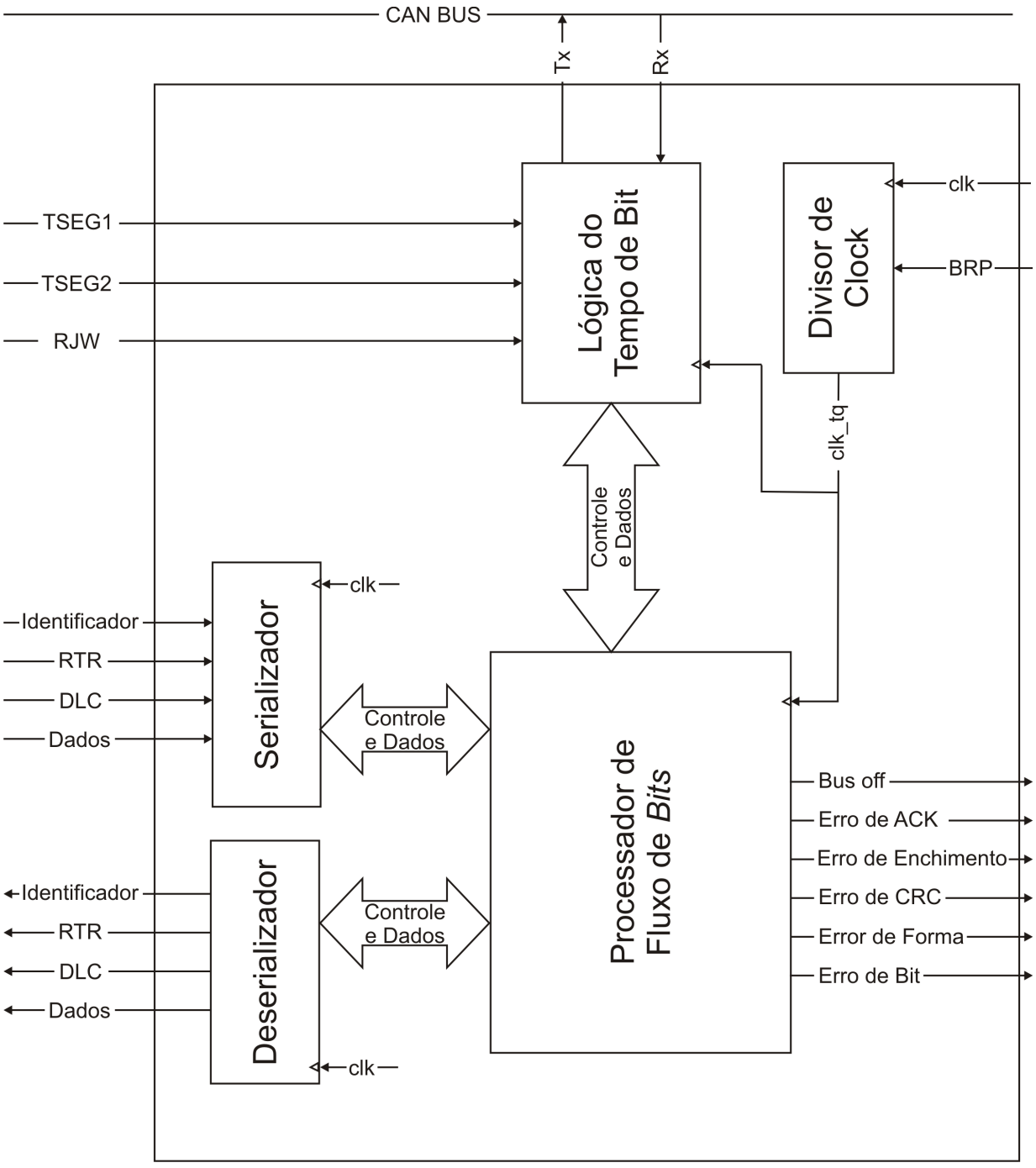

Figura 4.2 *– Módulos presentes no controlador CAN.*

$$clk_{tq} = 2 * clk(32 * BRP[4] + 16 * BRP[3] + 8 * BRP[2] + 4 * BRP[1] + 2 * BRP[0] + 1) \quad (4.1)$$

A leitura e a escrita do barramento são realizadas pela Lógica do Tempo de Bit. Para isso, ele implementa o Tempo do Bit, descrito na seção 3.3.4, e a sincronização, seção 3.3.5.

O Serializador recebe os dados necessários para o envio da mensagem, que são o identificador do nó que receberá a mensagem, o tamanho do dado, o RTR e o dado a ser transmitido. Assim, ele salva os dados em uma FIFO até que estes possam ser transmitidos. Quando o controlador está apto a transmitir uma mensagem, o módulo passa as sequências da FIFO bit a bit para o Processador de Fluxo de Bits.

O Deserializador realiza o processo inverso. Ele recebe os dados bit a bit do Processador de Fluxo de Bits, organiza e salva-os em uma FIFO. Assim como no Serializador, o Deserializador recebe os bits referentes ao identificador do nó que deve receber a mensagem, o tamanho do dado, o RTR e o dado transmitido.

O processo de criar todos os campos das mensagens a ser transmitidas, a recepção das mensagens e o controle lógico de todo o controlador são funções do Processador de Fluxo de Bits. Entretanto, como cada uma dessas funções exigi um alto grau de complexidade, a construção desse módulo seria difícil e tornaria o arquivo muito extenso. Portanto, ele foi subdividido em três outros módulos, como mostrado na Figura 4.3.

A criação de todas as mensagens descritas na seção 3.3.1 é implementada na Unidade de Transmissão, assim como a verificação dos erros que podem acontecer durante a transmissão. Na existência de algum erro, este é reportado ao módulo Inteligência Local. Na medida em que os bits fornecidos pelo Serializador são enviados, a Unidade de Transmissão vai criando internamente a sequência CRC a ser enviada.

Quando o controlador não está transmitindo, ele pode receber uma mensagem. A Unidade de Recepção é responsável pela recepção dos bits e por verificar a existência de erros. Caso não exista nenhum erro, o módulo envia o ACK *Slot* como dominante para o barramento e repassa os bits necessários ao Deserializador. Na ocorrência de um erro, este é reportado ao Inteligência Local.

O Inteligência Local controla todos os módulos do controlador, sendo o responsável por todos os sinais de controle. Além disso, ele gerencia a unidade de erro que determina qual o estado do controlador: se apresenta “erro passivo”, “erro recessivo” ou “*bus off*”. Como tanto a Unidade de Transmissão quanto a Unidade de Recepção podem escrever no barramento, o

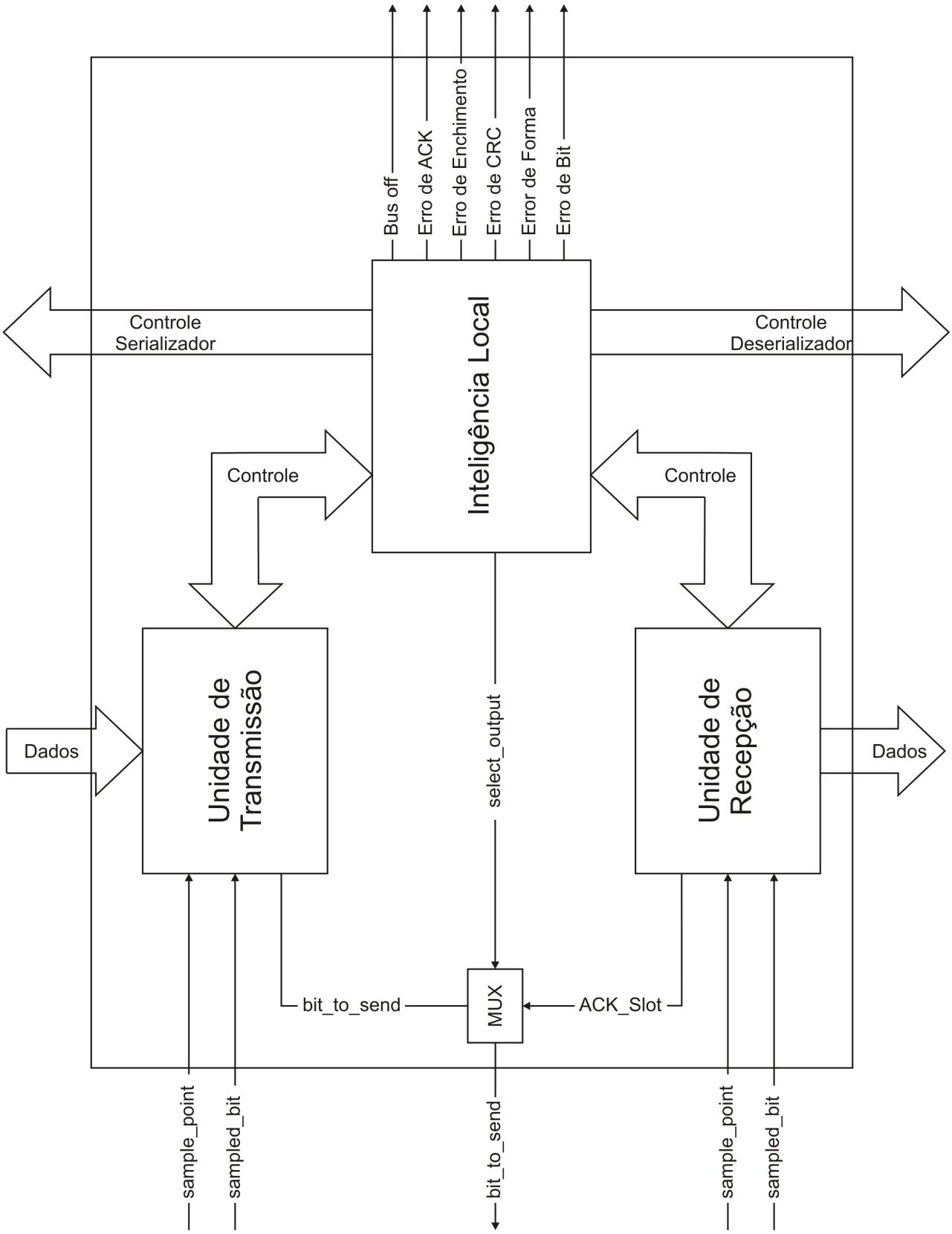

Figura 4.3 *– Subdivisão do processador de fluxo de bits.*

Inteligência Local controla qual módulo tem a permissão de enviar um bit ao barramento.

4.3 Simulação e verificação

O Xilinx ISE 12.3 fornece um conjunto de softwares auxiliares para desenvolvimento e depuração. Dentre estes softwares, o Xilinx ISim 12.3 é um simulador que auxilia na verificação da performance funcional e simulação de tempo para diversas linguagens. Assim, ele será utilizado na simulação da comunicação de dois nós CAN conectados ao mesmo barramento.

Como uma das vantagens da rede CAN é que os nós podem trabalhar em diferentes frequências, os controladores serão configurados para o primeiro operar em 100 MHz e o segundo, em 16 MHz. Estes valores foram escolhidos de forma a coincidir com os da implementação física. Fixando a velocidade da comunicação em 125Kbit/s, o registrador BRP deverá ser configurado em 0x18 e 0x03 para o primeiro e o segundo nó respectivamente. O número de ciclos de *clock* por bit deve ser o mesmo para ambos, assim os registradores TSEG1, TSEG2 e RJW deverão ser configurados em 0x0C, 0x01 e 0x01 respectivamente.

A simulação será dividida em três partes: comunicação simples, bit de enchimento e arbitragem. Na primeira, um nó enviará uma mensagem simples, sem bit de enchimento, para o segundo nó. Na segunda parte, a mensagem transmitida deve forçar a utilização do bit de enchimento. Na última parte ambos os nós tentarão transmitir ao mesmo tempo, sendo necessária a verificação da arbitragem para determinar qual dos nós irá transmitir a mensagem.

Na comunicação simples, o dado a ser transmitida será 0x8AE58AE58AE58AE5 com o identificador 0x4AB, uma vez que não possui mais de cinco bits consecutivos iguais em toda a mensagem. Em seguida, o controlador deverá transmitir 0xFF como dado e 0x482 como identificador, forçando a utilização de bit de enchimento. Para a arbitragem do barramento, o primeiro nó transmitirá com o identificador 0x13A e o segundo tentará transmitir uma mensagem com o identificador 0x13C; o dado não importa, uma vez que a arbitragem acontece antes e será enviada como 0x55.

4.4 Implementação em FPGA

A implementação física do sistema é importante para garantir que o controlador seja capaz de se comunicar com outro controlador. Dessa forma, o controlador CAN desenvolvido será implementado na placa de desenvolvimento XUPV5-LX110T da Xilinx e se comunicará com o USB-to-CAN compact da IXXAT, como mostra a Figura 4.4. Como este funciona a 16 MHz, o

controlador implementado no FPGA irá usar um *clock* de 100 MHz, e ambos serão configurados para trabalhar a 125 Kbit/s, de acordo com as configurações descritas na seção 4.3.

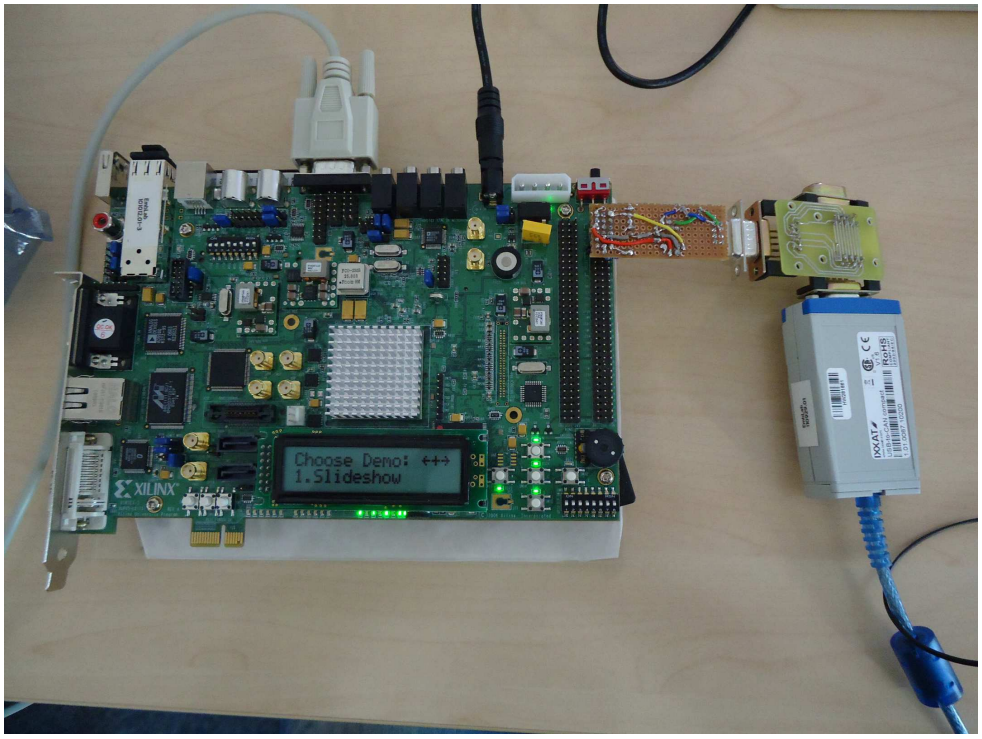

Figura 4.4 *– Implementação física do barramento CAN usando o controlador desenvolvido em FPGA.*

Para a validação do envio de mensagens do controlador, serão enviadas quatro mensagens diferentes. A primeira será uma mensagem simples que não forçará a utilização do bit de enchimento, com o dado 0x8AE58AE58AE58AE5 e identificador de 0x4AB. Em seguida, o controlador forçará a utilização do bit de enchimento com 0xFF como dado e 0x482 como identificador. Na terceira mensagem, o controlador irá solicitar o envio de uma mensagem através de um Mensagem Remota. A quarta e última será uma mensagem com o identificador 0x287 e o dado 0xBBCCDDEEFF.

O USB-to-CAN então enviara as mesmas mensagens para o controlador no FPGA. O número de mensagens deve ser superior ao valor máximo de mensagens suportado pela FIFO do Deserializador. Para verificar as mensagens recebidas no FPGA, os LEDs da placa deverão ser selecionados a partir das chaves para mostrar o identificador, os dados, o RTR, o tamanho da mensagem e os possíveis erros que o controlador pode detectar.

5 Resultados

Neste capítulo, é apresentada a lógica encontrada para o funcionamento do controlador (seção 5.1) e os resultados obtidos nos testes de simulação (seção 5.2) e implementação em FPGA (seção 5.3).

5.1 Lógica utilizada na implementação

O controlador CAN possui um módulo superior que engloba todos os módulos descritos neste trabalho. Ele fará a interface com o usuário.

A Figura 5.1 mostra o diagrama de blocos do Divisor de *Clock*. Esse módulo implementa um contador que incrementa um a cada borda de subida do *clock*. Esse contador é comparado com o Taxa de Divisão (BRP) para então inverter o valor do **clk_tq**.

O módulo Lógica do Tempo de Bit, como citado anteriormente na seção 4.2, deve realizar a leitura e escrita do barramento, ou seja, esse módulo deve implementar o tempo de bit descrito na seção 3.3.4. Para isso, foi utilizada a lógica do diagrama de bloco da Figura 5.2. Inicialmente, o módulo escreve o bit fornecido pelo Processador de Fluxo de Bits. Em seguida, ele incrementa o contador **cont_tseg1** e monitora o barramento, se ocorrer uma mudança de recessivo para dominante, o **phase_error** recebe o valor do **cont_tseg1** limitado ao RJW. Essa rotina é realizada até o valor do **cont_tseg1** ser igual à soma do TSEG1 com **phase_error**. Só então o módulo passa o valor do barramento para o Processador de Fluxo de Bits. Então é iniciada uma nova rotina que incrementa o contador **cont_tseg2** e monitora o barramento, atualizando o **phase_error** com o valor do contador limitado ao RJW na mudança do barramento. O processo é reiniciado quando **cont_tseg2** é igual à subtração do TSEG2 com **phase_error**.

O Processador de Fluxo de Bits recebe os dados fornecidos pelo Serializador para montar a palavra a ser transmitida. Nesse processo, é importante que o Identificador, o DLC, o RTR e os dados sejam guardados em uma FIFO com capacidade de armazenar até 16 sequências dessas. Como o Identificador e os dados são transmitidos bit a bit, são utilizados sinais de indicação da

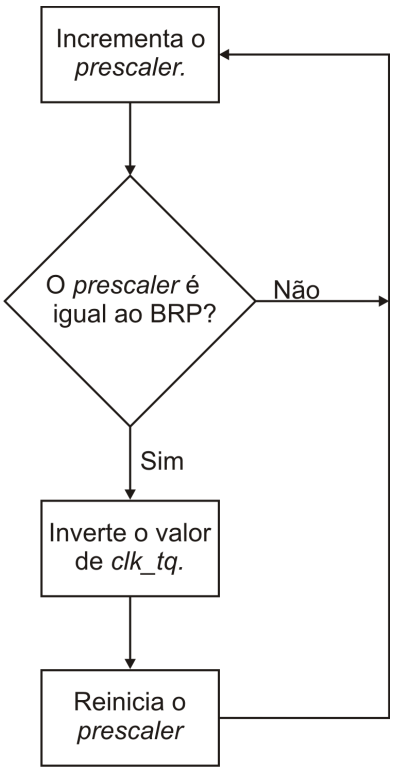

Figura 5.1 *– Diagrama de blocos do módulo Divisor de Clock.*

existência de um bit (*valid*) e o processador está pronto para ler o mesmo (*ready*), pois ambos trabalham com *clocks* diferentes. Existem ainda sinais que não necessitam ser enviados de forma serial: o DLC e o RTR, sendo enviados junto ao primeiro bit do Identificador.

Outro módulo que se comunica de forma semelhante com o Processador de Fluxo de Bits é o Deserializador. Entretanto, ele recebe o Identificador, o DLC, o RTR e os dados bit a bit e recompõe os *bytes*. Porém, para que estes sejam salvos, o Processador de Fluxo de Bits precisa da confirmação de que a mensagem foi recebida corretamente, o que ocorre quando a mensagem não possui nenhum erro.

Como mencionado, o Processador de Fluxo de Bits possui outros três módulos: Unidade de Transmissão, Unidade de Recepção e Inteligência Local. A única implementação necessária é o MUX controlado pelo Inteligência Local, para escolher qual bloco irá enviar o bit para o barramento: se a Unidade de Transmissão ou a Unidade de Recepção.

A Unidade de Transmissão é responsável pela transmissão das mensagens do controlador, tendo que decidir primeiro qual mensagem deve transmitir. Como mostrado na Figura 5.3, o módulo decide primeiro se irá transmitir uma mensagem de erro e de qual tipo será. Quando não é uma Mensagem de Erro, será transmitido o SOF e salvo o DLC e RTR no mesmo instante de *clock*. Em seguida, são enviados os bits do campo Identificador, verificando sempre a arbitragem, que, caso seja perdida, é sinalizado para o Inteligência Local. Após enviar o campo

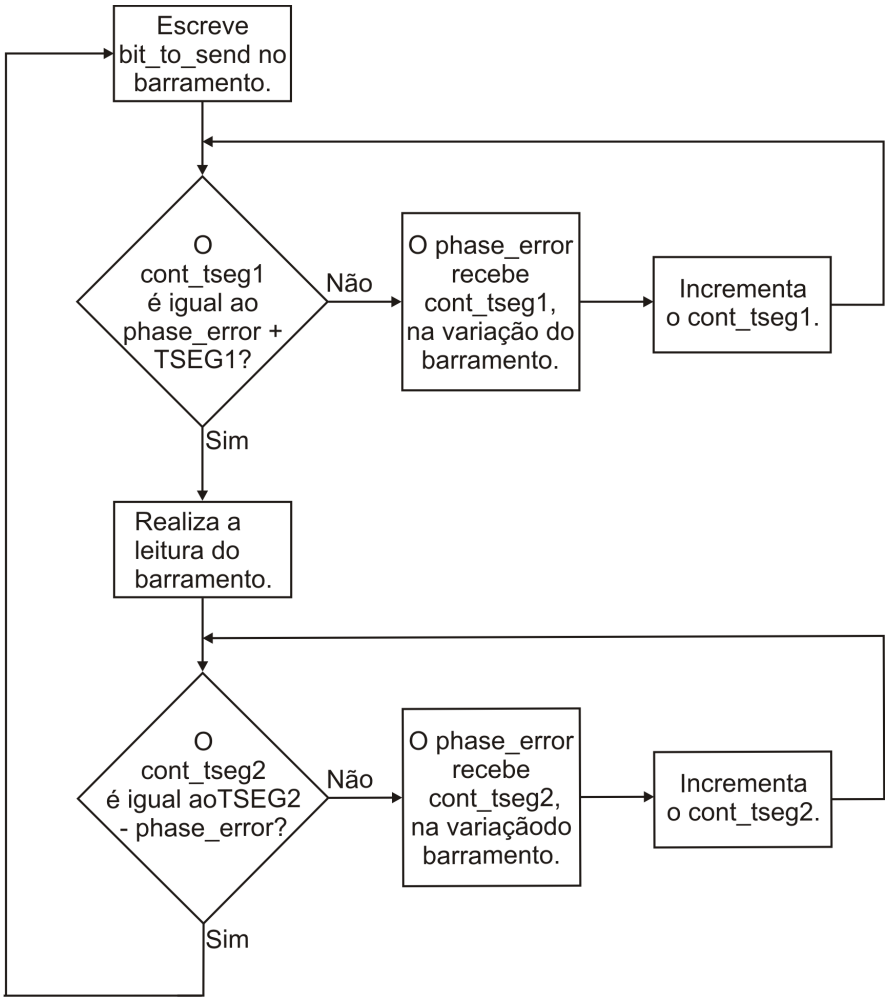

Figura 5.2 *– Diagrama de blocos do módulo Lógica do Tempo de Bit.*

de Controle, o controlador verifica o valor do RTR para decidir se transmite o campo de Dados, caso recessivo, ou caso dominante.

O cálculo da sequência CRC é realizado enquanto cada bit é transmitido, iniciando pelo SOF e terminando com o último bit do campo Dado, e então é enviado. Sua implementação é bastante simples, como pode ser visto na Figura 5.4. Ao receber um bit, é realizada uma operação XOR com esse bit e o bit mais significativo de registrador **crc_rg**, que contém a sequência CRC. Em seguida, o **crc_rg** é deslocado uma posição para a esquerda. Se o resultado da operação XOR for verdadeiro, uma nova operação XOR é realizada com o **crc_rg** e a sequência 0x4599, a representação normal em hexadecimal da Equação 3.1.

Para confirmar que a mensagem transmitida foi recebida, o controlador espera ler um bit dominante no barramento no campo ACK *Slot*. Quando é detectado um bit recessivo, o *flag* do Erro de ACK é sinalizado como nível lógico ativo. Se não ocorrerem erros, são enviados o campo EOF e, em seguida, o Espaço Entre Mensagens. Durante a transmissão de cada bit, o controlador insere o bit de enchimento e verifica se o bit do barramento é o mesmo que está sendo transmitido respeitando a regra de arbitragem. Em caso de erro, o *flag* de Erro de bit é sinalizado como nível lógico ativo.

Uma forma de proteger o barramento é remoção dos nós que apresentarem um número muito grande de erros. Para que o nó seja reinserindo no barramento, é preciso que seja detectada 128 vezes uma sequência de 11 bits recessivos. A verificação dessa sequência é realizada pela Unidade de Recepção antes de iniciar a recepção de qualquer mensagem, como pode ser visto na Figura 5.5. Se o controlador entra em estado *bus off*, esse módulo só volta a receber uma mensagem quando o Inteligência Local detecta as 128 sequências.

Após verificar que o controlador não se encontra em estado *bus off*, ele espera o início do SOF para começar a recepção da mensagem, como mostra a Figura 5.6. Durante a transmissão de uma mensagem, pode ocorrer uma disputa de arbitragem e, quando o nó perde essa disputa, ele deve receber a mensagem. Se o nó ganhar a arbitragem, o controlador para de receber a mensagem. Ou seja, o controlador recebe a mensagem pelo menos até o campo identificador mesmo quando está transmitindo. Ao verificar o estado do IDE, é detectado o formato da mensagem, padrão ou estendida, sendo esta última ignorada pelo controlador. Com o fim do campo de Controle, verifica-se a necessidade do recebimento ou não do campo Dado ao ler o RTR.

A CRC realizada pela Unidade de Recepção é idêntica à implementado na Unidade de Transmissão, descrita na Figura 5.4. Quando a sequência CRC recebida não é igual à calculada no controlador, o *flag* Erro de CRC é sinalizado como nível lógico ativo. Caso a CRC seja a

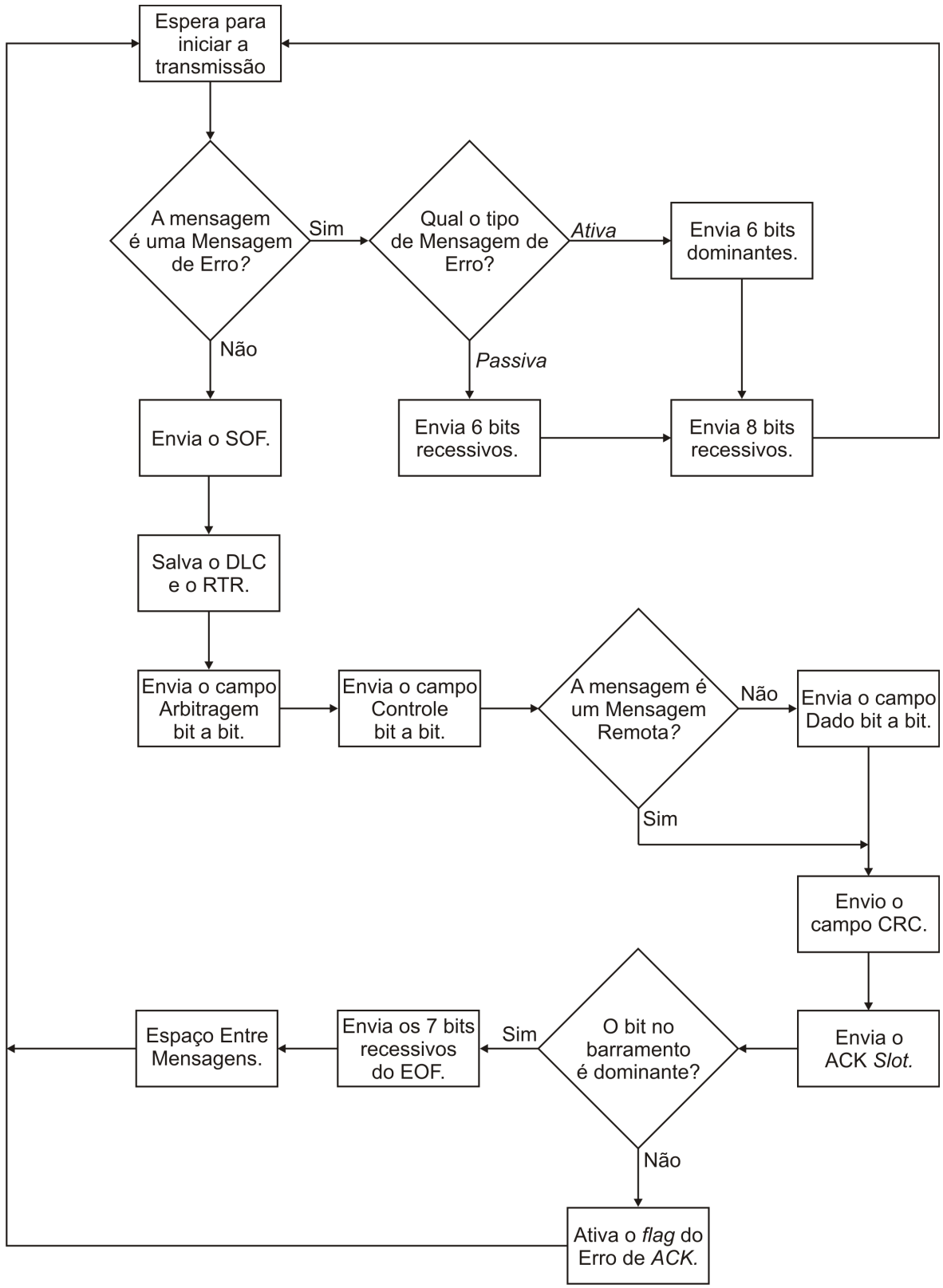

Figura 5.3 *– Diagrama de blocos do módulo Unidade de Transmissão.*

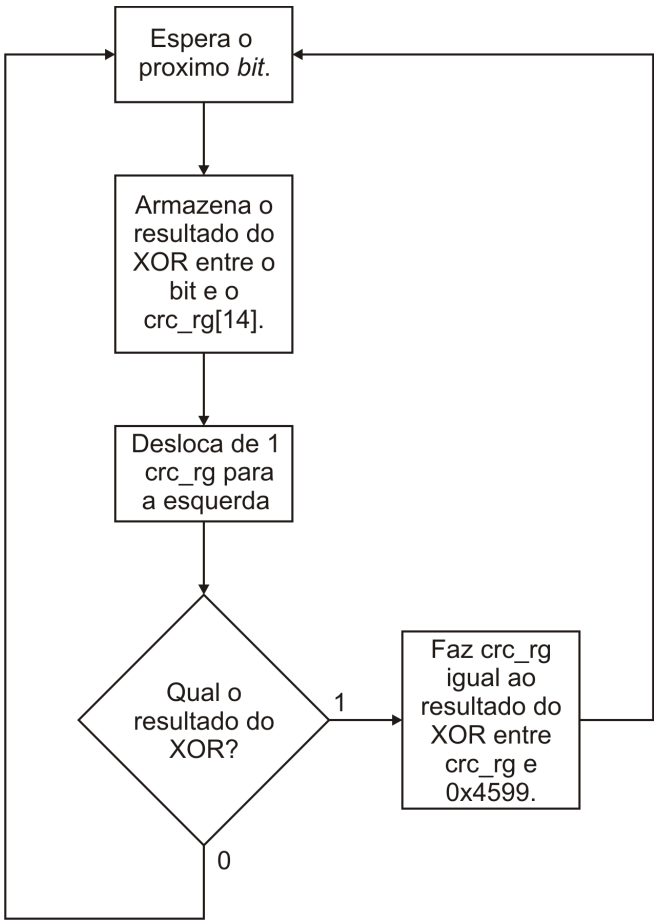

Figura 5.4 *– Diagrama de blocos do CRC.*

mesma, o controlador envia o ACK *Slot* como dominante e recebe o Delimitador do ACK e o campo EOF.

Durante toda a recepção, é realizada a remoção dos bits enchimento. Quando este não é apresentado, o *flag* Erro de Enchimento é sinalizado como nível lógico ativo, assim como a detecção de Erro de Forma resulta num nível lógico ativo do *flag* Erro de Forma.

A Inteligência Local determina quando o controlador irá transmitir ou receber uma mensagem, sendo responsável também por implementar os contadores de erro. Como mostra a Figura 5.7, antes de decidir qual decisão irá tomar, o módulo verifica a condição de *bus off*. Caso o controlador se encontre nesse estado, são esperadas 128 sequências de 11 bits recessivos para reativar a leitura e escrita no barramento. Quando o Serializador indica que possui uma mensagem para transmitir, o Inteligência Local habilita a transmissão à Unidade de Transmissão. Se a Unidade de Recepção detectar o início de uma transmissão no barramento, o Deserializador é preparado para o recebimento da mensagem e o MUX presente no Processador de Fluxo de Bits é ativado para que a transmissão de bits seja feita pela Unidade de Recepção.

Ao decidir pela transmissão de uma nova mensagem, o Inteligência Local habilita a Uni-

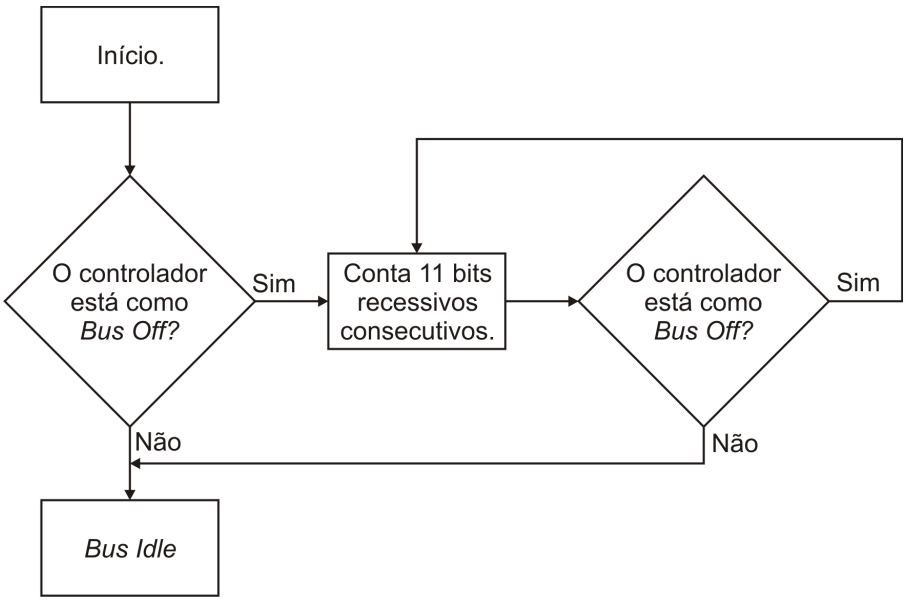

***Figura 5.5** – Primeira parte do diagrama de blocos do módulo Unidade de Recepção.*

dade de Transmissão e desabilita o bit de mensagem de erro. Como mostrado na Figura 5.8, ao ser notificado do resultado da arbitragem, o módulo ou continua a transmissão ou pula para a lógica de recebimento. Com a continuação, é esperado o fim da transmissão sem erro para voltar ao início da lógica do módulo. Quando ocorre um erro, o Serializador é sinalizado para que não perca os dados da mensagem e habilita o bit de mensagem de erro.

A Figura 5.9 mostra o diagrama de blocos da lógica de recebimento do módulo Inteligência Local. Ao habilitar o recebimento, o Deserializador inicia o recebimento dos bits para recompor as palavras. Não ocorrendo nenhum erro, a mensagem é validada e o Deserializador deve salvar a mensagem. Se ocorrer algum tipo erro, a Inteligência Local sinaliza para não salvar a mensagem e indica à Unidade de Transmissão que inicie a transmissão da mensagem de erro.

5.2 Simulação e Verificação

A simulação da transmissão de uma mensagem simples, proposta na seção 4.3, pode ser vista na Figura 5.10. O **identifier_transmitter**, **data_size_transmitter**, **data_transmitter** e **request_data_transmitter** representam o identificador, o DLC, o dado e o RTR, respectivamente, da mensagem a ser transmitida. Enquanto o **identifier_out_receiver**, **data_size_out_receiver**, **data_out_receiver** e **request_data_receiver** são o identificador, o DLC, o dado e o RTR, respectivamente, que o nó receptor leu do barramento. A máquina de estado do nó transmissor é a **transmitter_state**, enquanto a do nó receptor é a **receiver_state**. O barramento está nomeado como **CAN_BUS**.

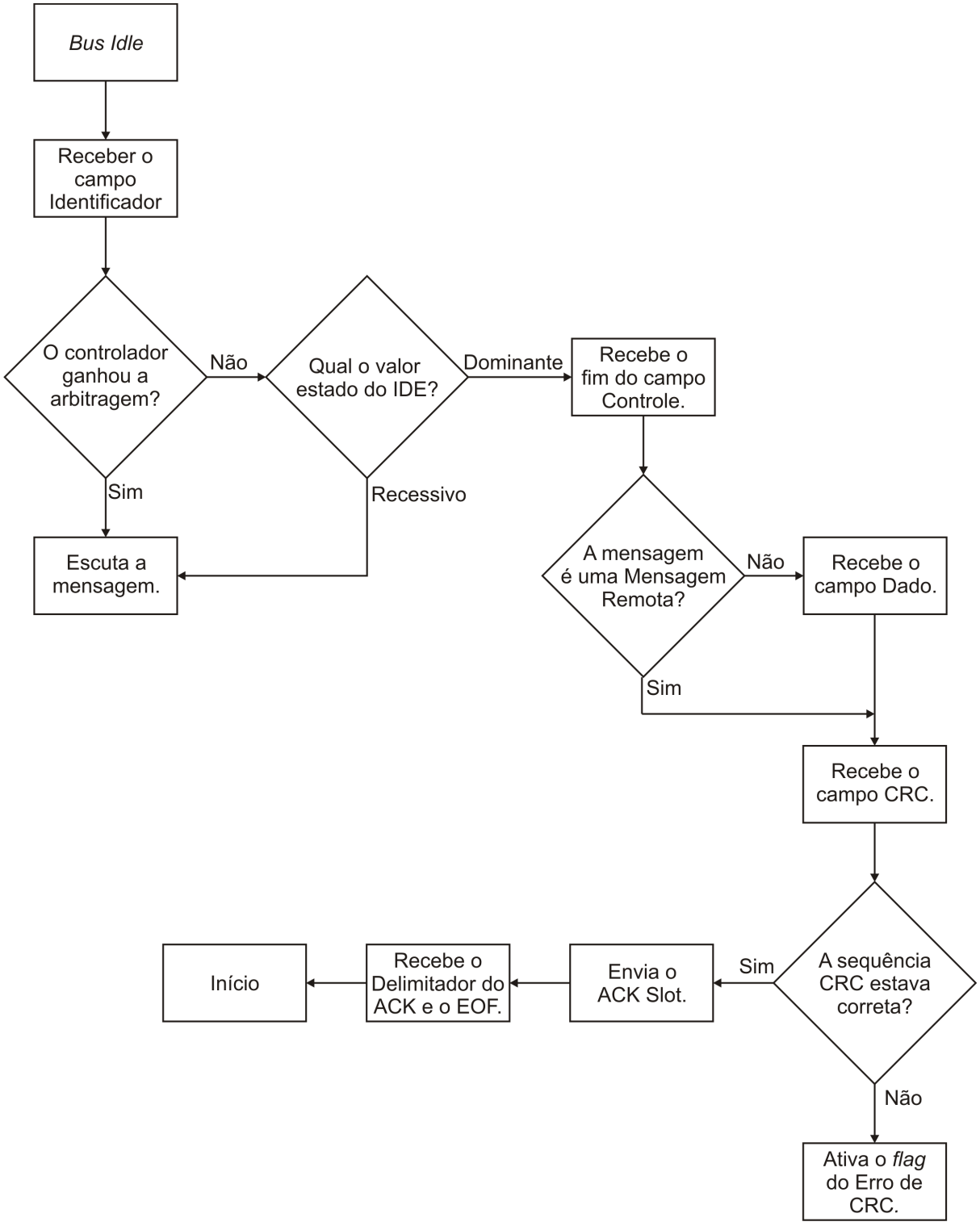

Figura 5.6 *– Segunda parte do diagrama de blocos do módulo Unidade de Recepção.*

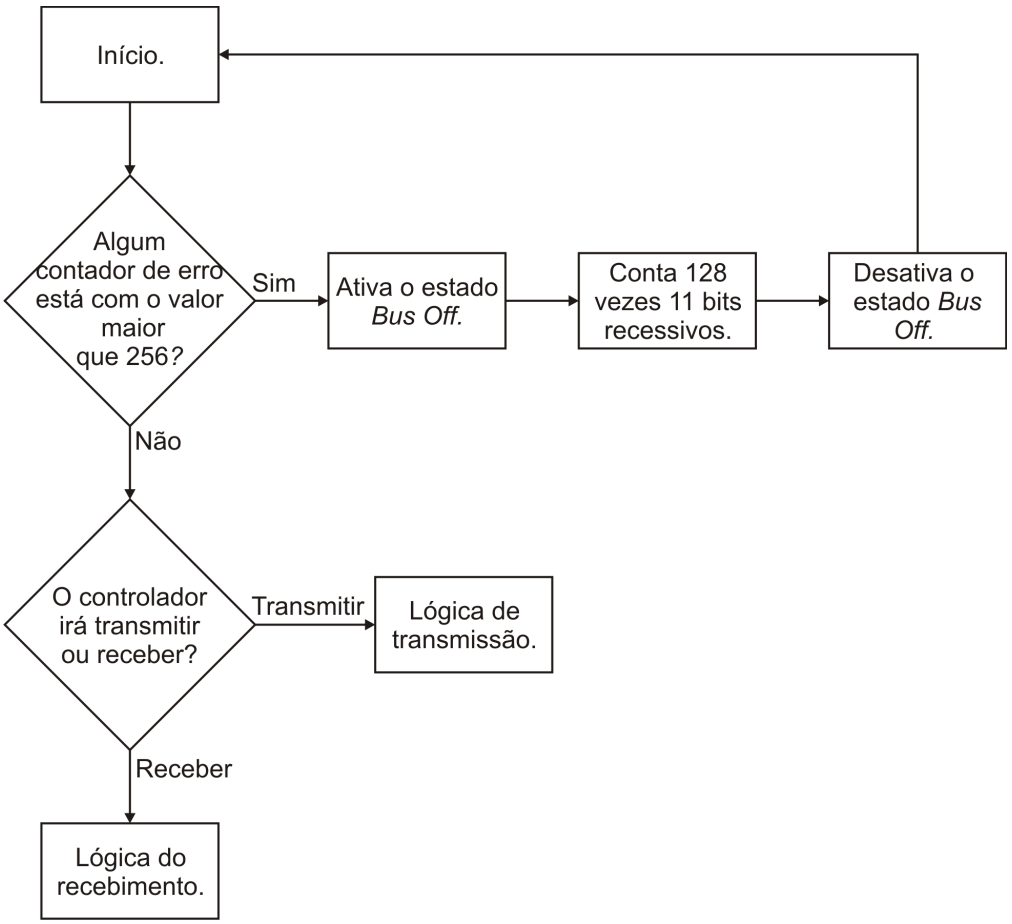

Figura 5.7 *– Diagrama de blocos do módulo Inteligência Local.*

O nó transmissor começará a transmissão ao escrever o SOF no início da simulação, Figura 5.10. Então, o nó receptor irá detectar o início de uma transmissão e aguardará o recebimento dos bits seguintes. O nó transmissor irá enviar os bits ao barramento até 800 μs, quando o receptor envia o ACK *Slot* como recessivo, indicando que a mensagem foi recebida. Ao final da mensagem, nenhum erro é detectado e então os dados recebidos são salvos nos registradores de saída do receptor, o que ocorre aos 840 μs.

A mensagem anterior não fazia uso do bit de enchimento, enquanto a simulação da Figura 5.11 utiliza um identificador e um dado que forçam a utilização desse recurso. Na figura, vê-se que o **stuff_bit** é ativado quando é detectada uma sequência de cinco bits, tanto no transmissor quanto no receptor. Ao ocorrer isso, o receptor desconsidera o próximo bit transmitido.

A simulação da arbitragem é feita usando dois nós conectados a um mesmo barramento CAN que iniciam a transmissão ao mesmo tempo, conforme ilustram as Figuras 5.12 e 5.13. O **indentifier_in 1** e o **indentifier_in 2** mostram os identificadores que os nós 1 e 2, respectivamente, desejam transmitir. O **transmitter_state** e o **receiver_state** representam as máquinas de estado da Unidade de Transmissão e da Unidade de Recepção respectivamente. Os identifi-

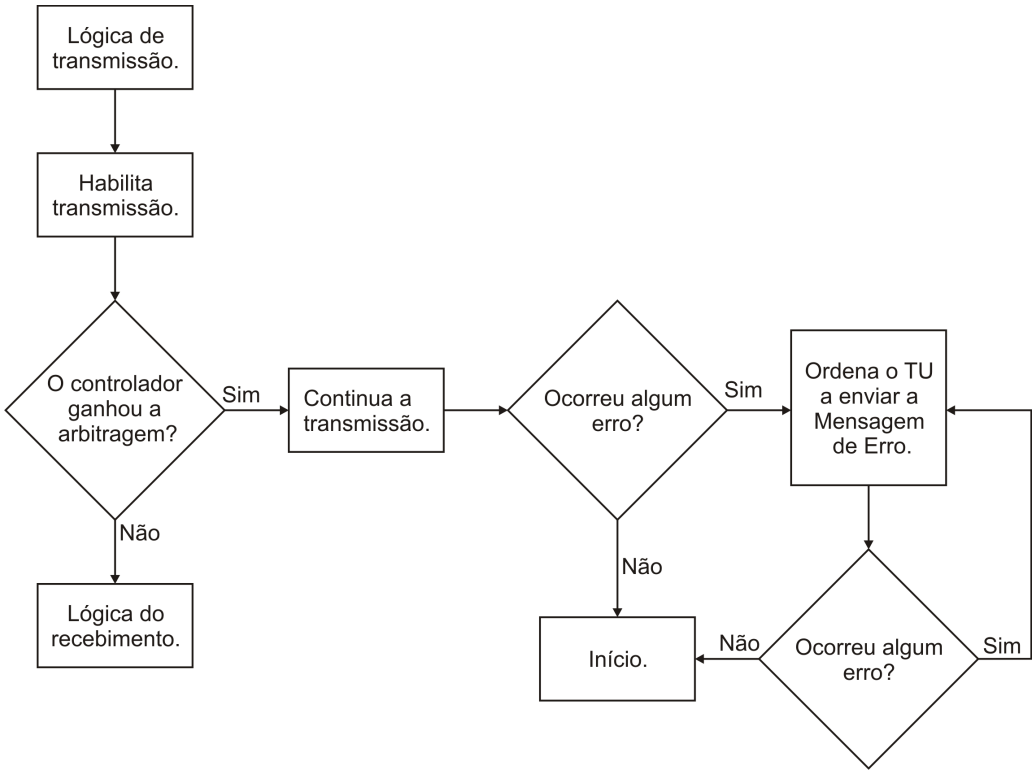

Figura 5.8 *– Diagrama de blocos da lógica de transmissão do módulo Inteligência Local.*

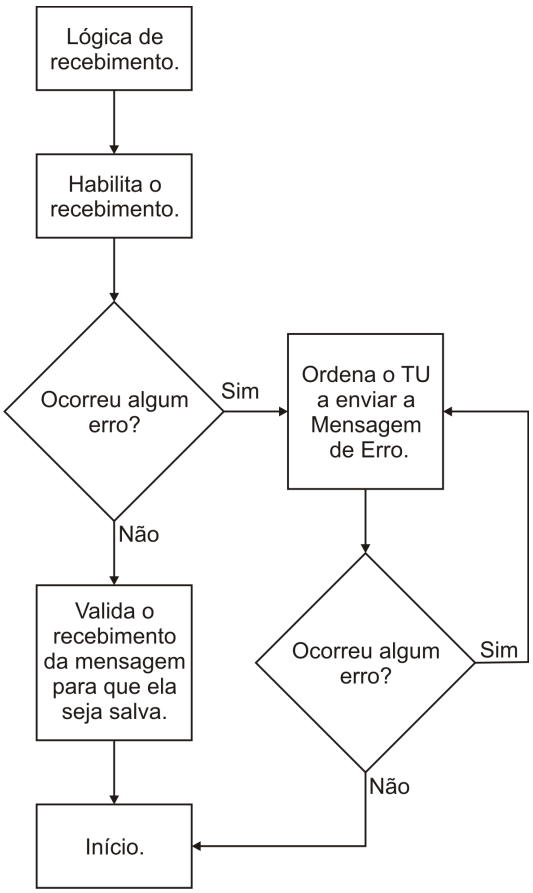

Figura 5.9 *– Diagrama de blocos da lógica de recebimento do módulo Inteligência Local.*

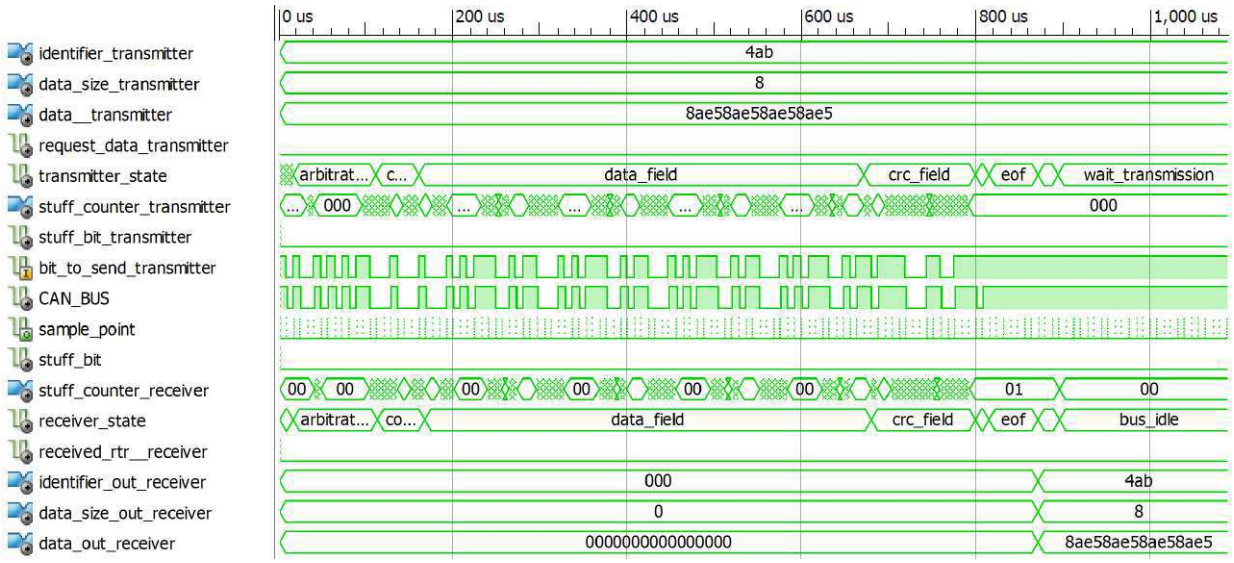

Figura 5.10 *– Simulação da transmissão de uma mensagem entre dois nós.*

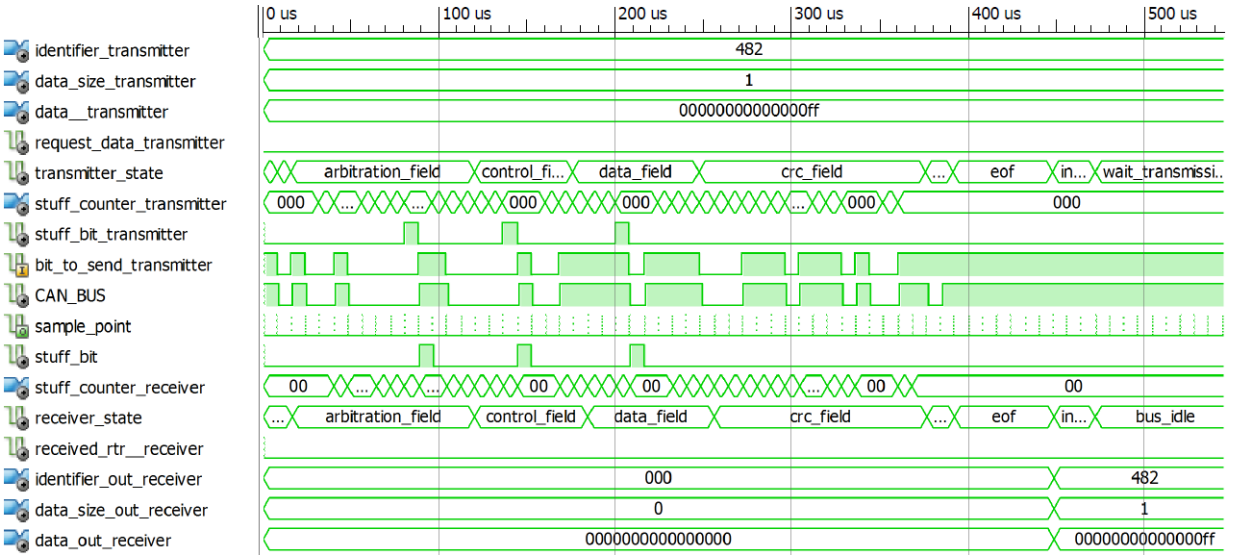

Figura 5.11 *– Simulação da transmissão de uma mensagem com bit de enchimento entre dois nós.*

cadores das mensagens recebidas são mostrados nos **indentifier_out**, onde o número indica da qual nó ele se refere. O barramento está representado pelo **CAN_BUS**.

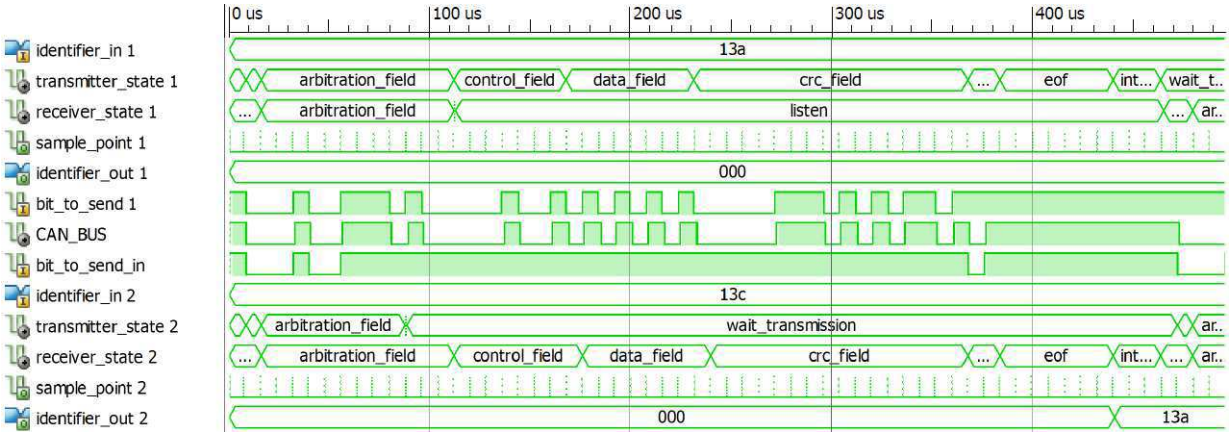

Figura 5.12 *– Simulação da disputa pela arbitragem entre dois nós entre 0 μs e 500 μs.*

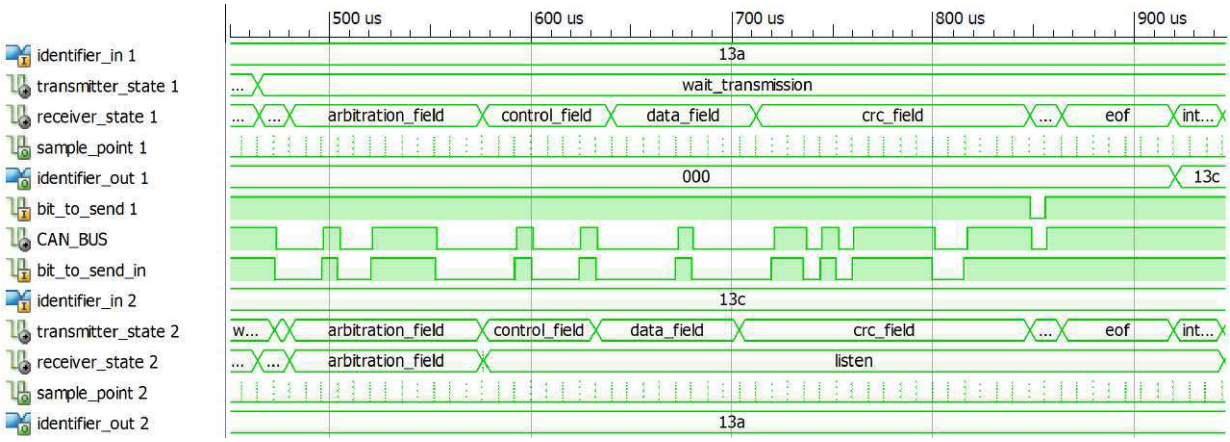

Figura 5.13 *– Simulação da disputa pela arbitragem entre dois nós entre 450 μs e 950 μs.*

Na Figura 5.12, pode-se notar que ambos os nós começam a transmitir no mesmo momento, $10\mu s$, quando o bit a ser enviado (**bit_to_send**) muda para recessivo e o barramento acompanha essa transmissão. Os dois comandam o barramento até 80 μs, quando o oitavo bit do campo Identificador é transmitido e o primeiro nó ganha a arbitragem. O segundo nó para a transmissão e passa a só receber a mensagem. Aos 110 μs, o primeiro nó ganha a arbitragem e a Unidade de Recepção passa a escutar o que acontece no barramento, pois todos os bits do campo Arbitragem foram enviados com sucesso. Como mostra a Figura 5.13, o segundo nó inicia a transmissão da sua mensagem após receber a mensagem do outro.

5.3 Implementação em FPGA

O controlador foi sintetizado utilizando o Xilinx ISE 12.3, o que gerou a tabela mostrada na Figura 5.14. Vê-se que foram utilizados 4% dos registradores, 5% das LUTs e 29% dos blocos de entrada e saída (em inglês *Input/Output Block*, IOB) presentes no *chip* Vertex 5 presente na placa XUPV5-LX110T. O software ainda gerou um clock máximo proposto de aproximadamente 327MHz.

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	3,245	69,120	4%	
Number used as Flip Flops	3,245			
Number of Slice LUTs	4,146	69,120	5%	
Number used as logic	4,146	69,120	5%	
Number using O6 output only	4,146			
Number of occupied Slices	1,224	17,280	7%	
Number of LUT Flip Flop pairs used	4,148			
Number with an unused Flip Flop	903	4,148	21%	
Number with an unused LUT	2	4,148	1%	
Number of fully used LUT-FF pairs	3,243	4,148	78%	
Number of unique control sets	22			
Number of slice register sites lost to control set restrictions	35	69,120	1%	
Number of bonded IOBs	188	640	29%	
Number of BUFG/BUFGCTRLs	2	32	6%	
Number used as BUFGs	2			
Average Fanout of Non-Clock Nets	6.29			

Figura 5.14 *– Tabela gerada pelo Xilinx ISE 12.3 indicando a utilização de espaço do controlo na placa.*

As quatro mensagens propostas na seção 4.4 para o envio e recebimento foram realizadas e são mostradas na Figura 5.15. As mensagens, que possuem a indicação **Self** na coluna **Flag**, são as enviadas pelo USB-to-CAN compact da IXXAT. Nota-se que nenhum erro é detectado no barramento, pois o **Error warning level** apresenta-se verde, indicando que todos os dados foram transmitidos e recebidos corretamente.

A Figura 5.16 mostra que o controlador desenvolvido consegue receber até 16 mensagens sem apresentar erro. Ao tentar enviar uma nova mensagem, Figura 5.17, o USB-to-CAN compact não recebe o ACK *Slot*; envia uma mensagem de erro, indicado pelo **Error warning level** que apresenta-se vermelho; e tenta retransmitir a mensagem, mostrado pelo *Transmit pending* em vermelho. Ao executar a leitura da FIFO controlador na FPGA, a mensagem pendente é

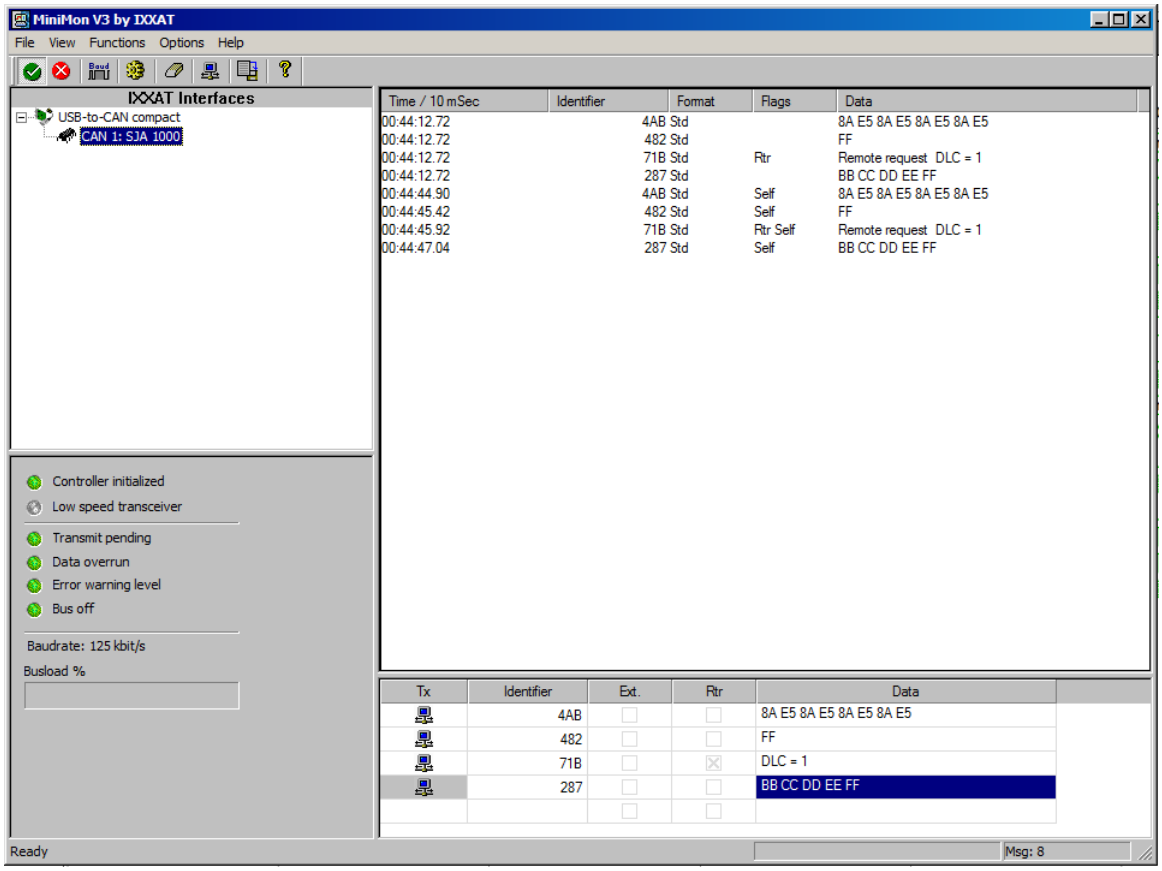

Figura 5.15 *– Janela do CAN MiniMonitor V3 com a transmissão e o recebimento de mensagem.*

então transmitida, como mostra a Figura 5.18.

Durante os testes, o controlador da IXXAT foi desativado via *software* e o controlador implementado na FPGA tentou enviar as mensagens ao barramento. Como não existia nenhum outro nó para receber as mensagens, verificou-se a ocorrência do Erro de Reconhecimento e a transmissão da mensagem de erro. Após 32 tentativas de transmissão, o controlador entrou em estado de "*bus off*" e suspendeu a transmissão de mensagem.

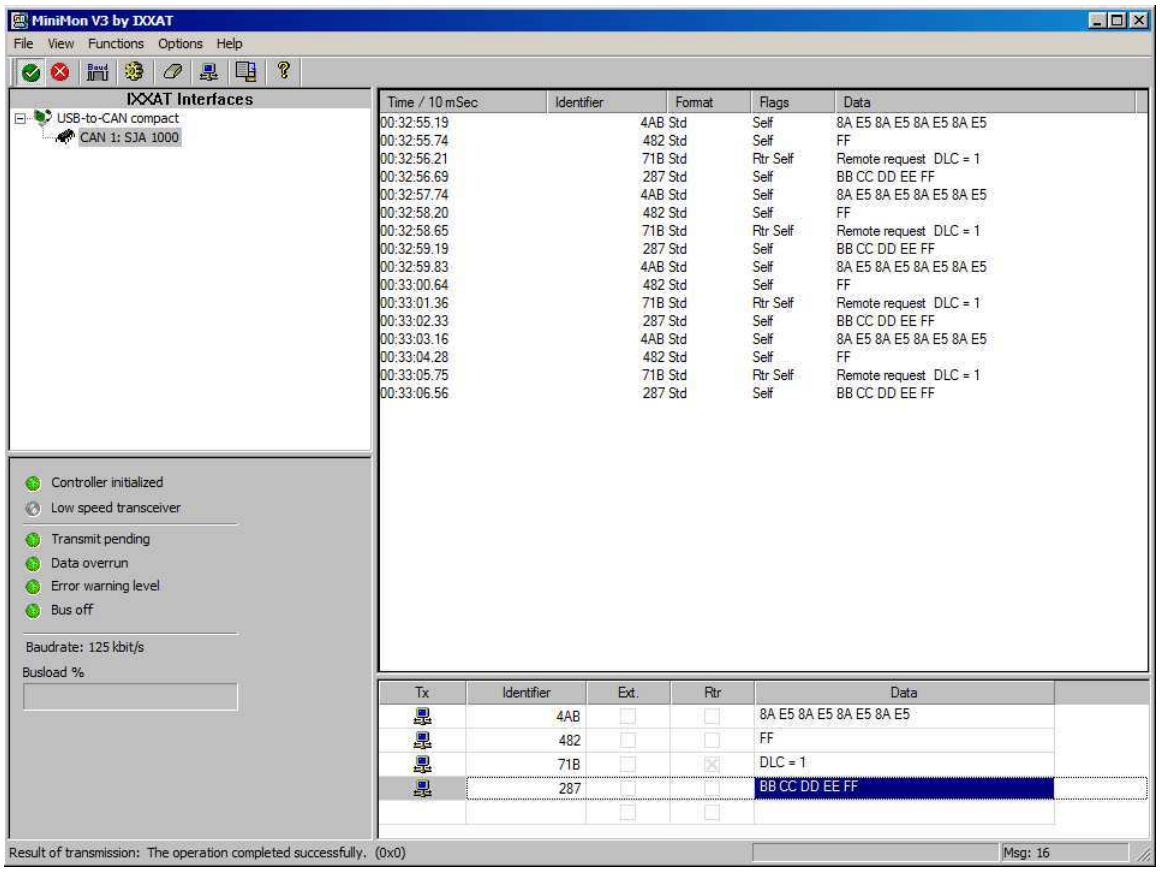

Figura 5.16 *– Janela do CAN MiniMonitor V3 mostrando a transmissão das 16 mensagens.*

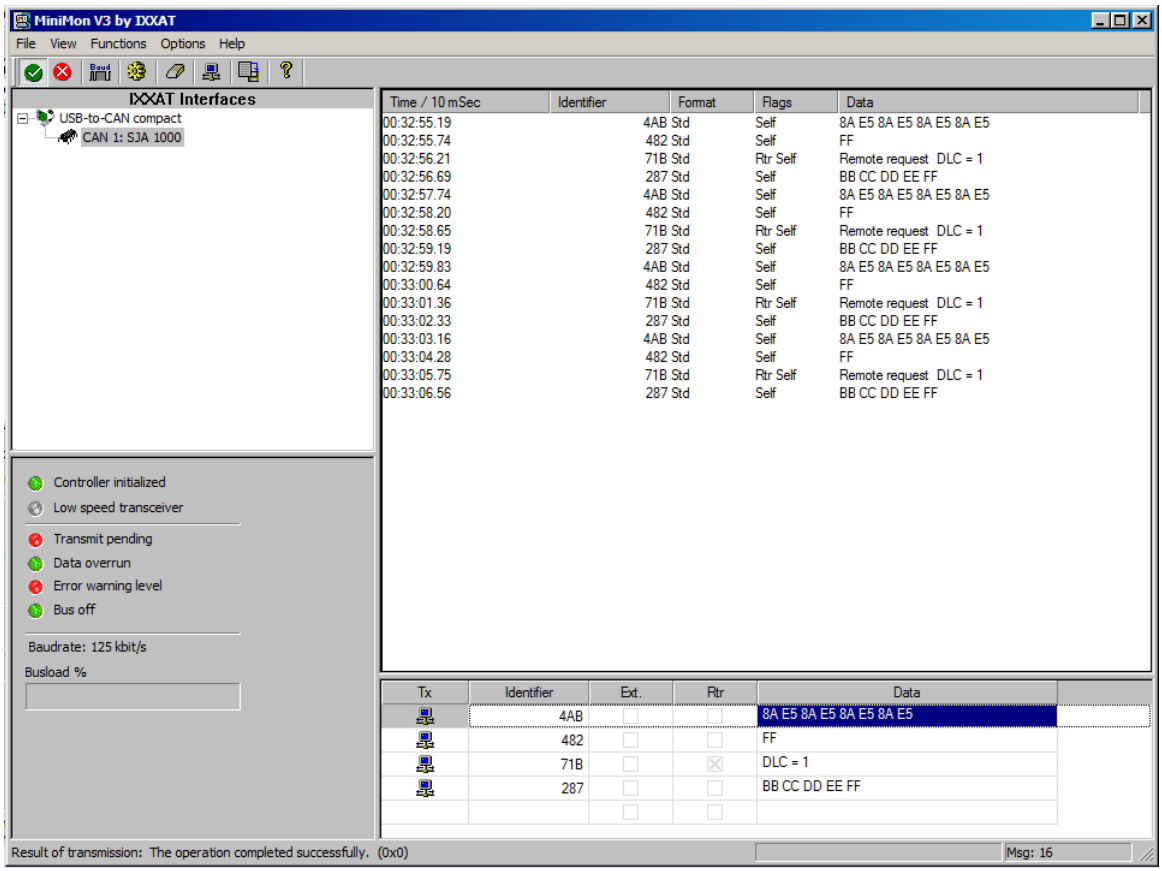

Figura 5.17 *– Janela do CAN MiniMonitor V3 indicando que a décima sétima mensagem não foi transmitida.*

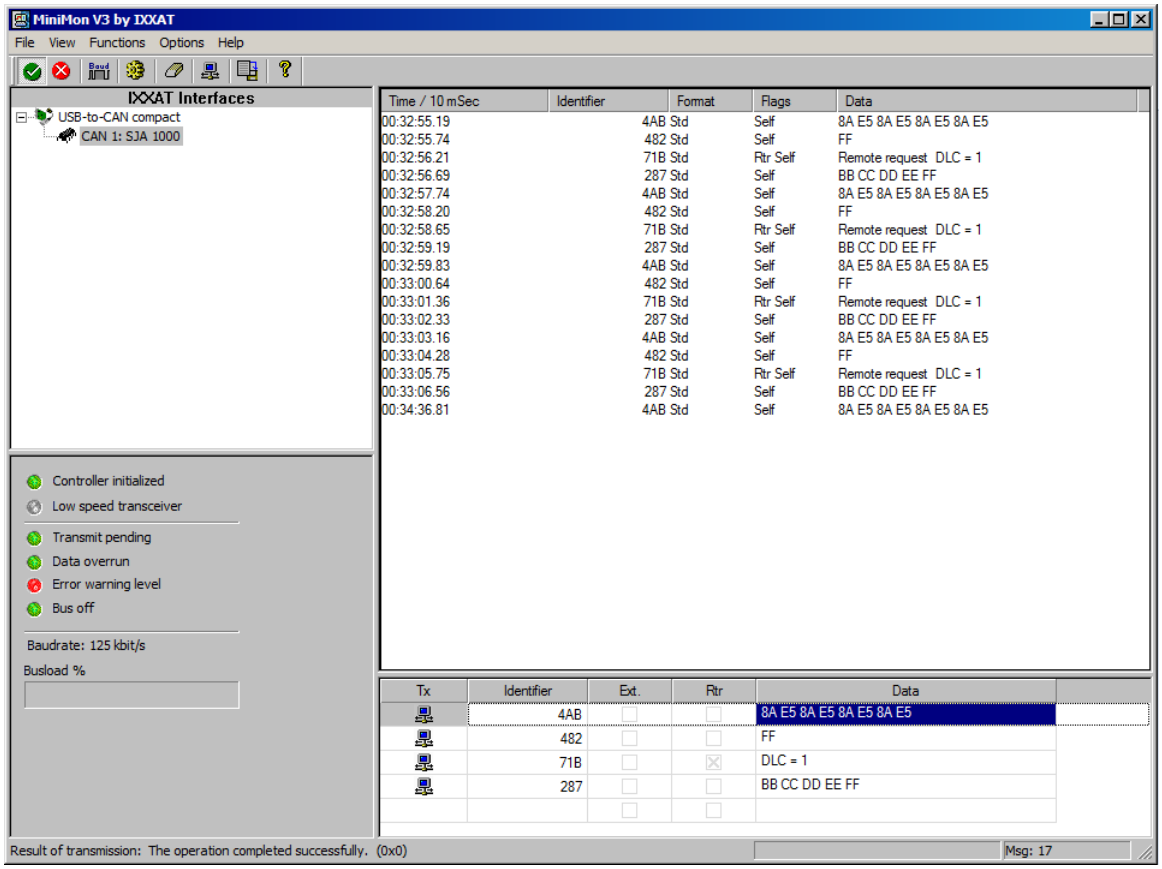

Figura 5.18 *– Janela do CAN MiniMonitor V3 indicando que a décima sétima mensagem foi transmitida.*

6 Conclusão

Neste trabalho, foi desenvolvido um controlador CAN usando a linguagem VHDL, baseado no proposto por (HARTWICH & BASSEMI 1999). Com intuito de utilizar o controlador em diversos sistemas, para formar as mensagens que serão transmitidas são solicitadas apenas informações básicas do usuário. O controlador é capaz de codificar e decodificar mensagens, requisitar uma mensagem, realizar controle de erro e arbitragem, atingindo os requisitos funcionais propostos.

O código final, incluindo todos os módulos, possui 3294 linhas. Usando cerca de 4% da placa XUPV5-LX110T, permite que outras ferramentas também sejam implementadas na mesma. Assim, os sistemas que estão em desenvolvimento podem utilizar o controlador para realizar seus testes se comunicando com outros.

Nos testes realizados, o controlador se comportou de forma semelhante tanto na simulação quanto na implementação em FPGA. As mensagens enviadas e recebidas não possuíram erros de codificação e decodificação, respectivamente. As mensagens de erro e a proteção para o barramento, o estado "*bus off*", apresentaram-se de forma adequada.

Como trabalhos futuros, pode-se adaptar o controlador para transmitir e receber mensagens no formato estendido. Para consolidar o funcionamento do controlador, pode-se realizar a verificação funcional proposta por Robert Bosch GmbH 1999, que não foi realizada neste trabalho devido ao seu alto custo de aquisição.

Referências Bibliográficas

BOURDON, P. R. G.; DELAPLACE, S. Can for autonomous mobile robot. **3rd international CAN Conference**, França, 1996. 1

CAMPBELL, J. **C programmer's guide to serial communications**. Indianapolis, IN, USA: Sams, 1987. ISBN 0-672-22584-0. 3.2

CIA. **CAN protocol**. CAN in Automation, Outubro 2011. Disponível em: <http://www.can-cia.org/index.php?id=systemdesign-can-protocol>. (document), 1, 3.1, 3.1, 3.2, 3.2

DAVIS ALAN BURNS, R. J. B. R. I.; LUKKIEN, J. J. Controller area network (can) schedulability analysis: Refuted, revisited and revised. **REAL-TIME SYSTEMS**, v. 35, n. 3, p. 239–272, 2007. 1, 3.1

FREDRIKSSON, L.-B. **Controller Area Networks - And the protocol CAN for machine control systems**. Kvaser, Janeiro 2012. Disponível em: <http://www.kvaser.com/sv/om-can/mer-information/78.html>. 1, 3.1

HARTWICH, F.; BASSEMI, A. The configuration of the can bit timing. **Robert Bosch GmbH Proceedings 6th International CAN Conference**, Turin, 1999. 3.3.4, 4.2, 6

HEFFERNAN, D.; LEEN, G. Expanding automotive electronic systems. **IEEE Computer**, v. 35, p. 88–93, 2002. 1, 3.1

IEEE. **IEEE Standard VHDL Language Reference Manual**. , 2009. . 4

INTERNATIONAL STANDARDS ORGANISATION. **ISO 11898: Road vehicles – Interchange of digital information – Controller area network (CAN) for high-speed communication**. [S.l.], 1993. 3.2

IXXAT. **USB-to-CAN Interface**. Novembro 2011. Disponível em: <http://www.ixxat.com/usb-to-can-compact-interface_en.html>. 4.1

KOOPMAN, P. 32-bit cyclic redundancy codes for internet applications. In: **Proceedings of the 2002 International Conference on Dependable Systems and Networks**. Washington, DC, USA: IEEE Computer Society, 2002. (DSN '02), p. 459–472. ISBN 0-7695-1597-5. 3.3.1, 3.3.3

MORAES, A. M. A. F. G.; JUNIOR, J. P. Sistema integrado e multiplataforma para controle remoto de residências. **VII Workshop Iberchip**, Uruguai, 2001. 1

ROBERT BOSCH GMBH. **CAN Specification - Version 2.0**. Stuttgart, 1991. 3.2, 3.3.1

ROBERT BOSCH GMBH. **VHDL Reference CAN - Users Manual**. Stuttgart, 1999. 6

SA, P. R. B. Jadsonlee da S.; NETO, J. S. da R. Implementação e analise de uma rede can para controle de um sistema distribuıdo. **5 Congresso Internacional de Automação, Sistemas e Instrumentação**, 2005. 3.2

SANTOS, F. S. **Reestruturação do ipPROCESS e Inclusão dos Processos Fundamentais do Ciclo de Vida**. Dissertação (Mestrado) — Universidade Federal da Pernambuco, Recife, Fevereiro 2009. . 1

SEMICONDUCTORS, P. **PCA82C250 data sheet**. , 2000. . 4.1

SOFTING. **CAN bus (Controller Area Network), an overview**. CAN in Automation, Novembro 2011. Disponível em: <http://www.softing.com/home/en/industrial-automation/products/can-bus/more-can-bus/index.php?navanchor=3010320>. 3.3.1, 3.3.4

WIKIPEDIA. **VHDL**. Fevereiro 2012. Disponível em: <http://en.wikipedia.org/wiki/VHDL>. 4

XILINX. **Xilinx University Program XUPV5-LX110T Development System**. Novembro 2011. Disponível em: <http://www.xilinx.com/univ/xupv5-lx110t.htm>. (document), 4.1