[[1]](#footnote-0)

Análise, Modelagem e Implementação em Verilog de um Decodificador de Frames do Protocolo CAN 2.0

David Alain do Nascimento, Aluno, CIn/UFPE, e Luís Felipe Prado D’Andrada, Aluno, CIn/UFPE

*Abstract*—These instructions give you guidelines for preparing papers for IEEE Transactions and Journals*.* Use this document as a template if you are using Microsoft *Word* 6.0 or later. Otherwise, use this document as an instruction set. The electronic file of your paper will be formatted further at IEEE. Paper titles should be written in uppercase and lowercase letters, not all uppercase. Avoid writing long formulas with subscripts in the title; short formulas that identify the elements are fine (e.g., "Nd–Fe–B"). Do not write “(Invited)” in the title. Full names of authors are preferred in the author field, but are not required. Put a space between authors’ initials. The abstract must be a concise yet comprehensive reflection of what is in your article. In particular, the abstract must be self-contained, without abbreviations, footnotes, or references. It should be a microcosm of the full article. The abstract must be between 150–250 words. Be sure that you adhere to these limits; otherwise, you will need to edit your abstract accordingly. The abstract must be written as one paragraph, and should not contain displayed mathematical equations or tabular material. The abstract should include three or four different keywords or phrases, as this will help readers to find it. It is important to avoid over-repetition of such phrases as this can result in a page being rejected by search engines. Ensure that your abstract reads well and is grammatically correct.

*Index Terms*—CAN, Controller Area Network, Verilog, FPGA, Máquina de estados.

# INTRODUÇÃO

C

AN (Controller Area Network) é um protocolo de comunicação desenvolvido pela Robert Bosch GmbH e lançado em 1986 [REF], e é principalmente utilizado em carros em sistemas do tipo Safety-Critical. No CAN, os pacotes (*frame*s) são transmitidos através de um barramento de dados com cabeamento de apenas dois fios [REF]. E, por utilizar uma topologia de barramento, poderá haver colisão de *frame*s quando mais de um nó da rede CAN transmitir simultaneamente. Para contornar este problema, o CAN implementa o protocolo *Carrier Sense Multiple Access Collision Resolution* (CSMA/CR) que soluciona o problema da colisão de *frame*s de maneira determinística, em que durante a arbitração ‘vence’, ou seja, é transmitido por completo, apenas o *frame* que possui a maior prioridade e os outros *frame*s param de ser transmitidos. O *frame* de maior prioridade é o que possuir o campo de arbitração com menor valor, em outras palavras, é o *frame* que possuir mais bits dominantes (valor 0) no campo de arbitração, comparando-se bit a bit da esquerda para direita [REF]. [http://www.eecs.umich.edu/courses/eecs461/doc/CAN\_notes.pdf]

O CAN possui diferentes versões: 1.0, 1.1, 1.2, 2.0 e CAN-FD [REF]. A versão 2.0, lançada em 1991 [REF], possui duas especificações: 2.0A, para compatibilidade com as versões anteriores 1.x e a versão 2.0B, que possui como principal diferença o identificador de 29 bits [REF].

O CAN utiliza o *Non-Return to Zero* (NRZ) como mecanismo de codificação na transmissão dos bits no barramento, e por conta disso, podem acontecer erros de sincronismo dos bits após vários bits de mesmo valor serem transmitidos. Para solucionar este problema, o CAN implementa um mecanismo chamado *Bit Stuffing*, em que após cinco bits sequenciais de mesmo valor, é inserido um bit de valor oposto após o quinto bit sequencial.

# Frames CAN 2.0

A versão 2.0 possui quatro diferentes tipos de *frame*s: *Data Frame*, *Remote Frame*, *Error Frame* e *Overload Frame* [REF]. Este projeto demonstra uma modelagem e implementação de um decodificador de *frame*s CAN 2.0 que reconhece os quatro diferentes tipos de *frame*s, como também os valores transmitidos entre os *frame*s, chamado de *Interframe Spacing*.

## Data Frame

O *Data Frame* pode ter dois formatos: o Base Frame, que possui identificador de 11 bits e o Extended Frame, que possui identificador de 29 bits.

### Base Frame

O Base Frame é composto pelos respectivos campos: SOF (*Start of Frame*), Identifier, RTR (Remote Transmission Request), IDE (Identifier Extension bit), Reserved Bit, DLC (Data Length Code), Data Field, CRC (Cyclic Redundancy Check), CRC Delimiter, ACK Slot, ACK Delimiter e EOF (*End of Frame*).

#### SOF (Start of Frame)

* Descrição: O SOF determina o início de um *frame* de dados.
* Tamanho do campo: 1 bit.
* Valores esperados: O único valor possível para o *Start of Frame* é dominante (0).

#### Identifier

* Descrição: Um identificador único para a mensagem a ser enviada. Seu valor é utilizado na arbitração e, por isso, é responsável pela prioridade de mensagens.
* Tamanho do campo: 11 bits.
* Valores esperados: Qualquer valor é aceitável, porém os 7 bits mais significativos não podem ser todos recessivos (1).

#### RTR (Remote Transmission Request)

* Descrição: Campo que identifica se uma mensagem é um *Data Frame* ou um *Remote Frame*.
* Tamanho do campo: 1 bit.
* Valores esperados: É utilizado o valor dominante (0) para um *Data Frame* e valor recessivo (1) para um *Remote Frame*.

#### IDE (Identifier Extension Bit)

* Descrição: Identifica o tipo de *frame* de dados, *Base Frame* ou *Extended Frame*.
* Tamanho do campo: 1 bit.
* Valores esperados: Para um *Base Frame*, o valor deste campo deve ser sempre dominante (0). Caso contrário, o *frame* será um *Extended Frame*.

#### Reserved Bit

* Descrição: Bit reservado para uso futuro.
* Tamanho do campo: 1 bit.
* Valores esperados: Deve ser dominante (0), porém, caso um receptor receba um bit recessivo (1), não deve ser gerado nenhum tipo de erro.

#### DLC (Data Length Code)

* Descrição: Define a quantidade de dados que serão enviados no campo *Data Field*, de forma que o valor recebido neste campo é igual a quantidade de bytes que serão recebidos no campo *Data Field*.
* Tamanho do campo: 4 bits.
* Valores esperados: São esperados valores de 0 a 8. Como o tamanho do campo permite o envio de valores até 15, caso um receptor receba um valor maior que 8, deve ser considerado apenas 8 e não deve ser gerado nenhum tipo de erro.

#### Data Field

* Descrição: Dados da mensagem.
* Tamanho do campo: É definido pelo valor recebido no campo. Pode variar de 0 a 64 bits.
* Valores esperados: Não há restrição para os valores a serem enviados no campo de dados.

#### CRC (Cyclic Redundancy Check)

* Descrição: Utilizado para identificar erros na transmissão dos dados.
* Tamanho do campo: 15 bits.
* Valores esperados: É esperado o valor resultante do cálculo de CRC. Um valor diferente deste levará a um CRC ERROR, que deve ser enviado apenas após o ACK Delimiter.

#### CRC Delimiter

* Descrição: É o bit seguinte ao CRC.
* Tamanho do campo: 1 bit.
* Valores esperados: É esperado um valor recessivo (1) para o CRC Delimiter. Um valor dominante (0) deve levar a um *Form Error*.

#### ACK Slot

* Descrição: Ao receber uma mensagem corretamente, um receptor deve enviar, através deste campo, sua confirmação de recebimento.
* Tamanho do campo: 1 bit.
* Valores esperados: Quando um receptor recebe uma mensagem corretamente o valor esperado no barramento é dominante (0). Quando nenhum receptor responde, ou seja, deixa o barramento em recessivo (1), o transmissor irá emitir um *ACK Error*.

#### ACK Delimiter

* Descrição: É o bit seguinte ao *ACK Slot*. Faz com que o *ACK Slot* seja cercado de bits recessivos.
* Tamanho do campo: 1 bit.
* Valores esperados: É esperado um valor recessivo (1) para o *ACK Delimiter*. Um valor dominante (0) deve levar a um *Form Error*.

#### EOF (End of Frame)

* Descrição: Determina o fim do *frame* de dados.
* Tamanho do campo: 7 bits.
* Valores esperados: Todos os bits devem ser recessivos (1), porém, para receptores, o último bit deve ser tratado como don’t care, ou seja, pode ser tanto dominante (0) quanto recessivo (1). Um valor diferente do esperado deve levar a um Form Error.

[https://www-inst.eecs.berkeley.edu/~ee249/fa08/Lectures/handout\_canbus2.pdf]

### Extended Frame

O Extended Frame é composto pelos respectivos campos: SOF (*Start of Frame*), Identifier A, SRR (Substitute remote request), IDE (Identifier Extension bit), Identifier B, RTR (Remote transmission request), Reserved Bits, DLC (Data Length Code), Data Field, CRC (Cyclic Redundancy Check), CRC Delimiter, ACK Slot, ACK Delimiter e EOF (*End of Frame*).

#### SOF (Start of Frame)

* Descrição: O SOF determina o início de um *frame* de dados.
* Tamanho do campo: 1 bit.
* Valores esperados: O único valor possível para o *Start of Frame* é dominante (0).

#### Identifier A

* Descrição: A primeira parte do identificador único para a mensagem a ser enviada. Seu valor é utilizado na arbitração e, por isso, é responsável pela prioridade de mensagens.
* Tamanho do campo: 11 bits.
* Valores esperados: Qualquer valor é aceitável, porém os 7 bits mais significativos não podem ser todos recessivos (1).

#### SRR (Substitute Remote Request)

* Descrição: Campo que substitui o RTR, em termos de localização no *frame*, no formato do Extended Frame.
* Tamanho do campo: 1 bit.
* Valores esperados: Deve ser recessivo (1), porém, caso um receptor receba um bit dominante (0), não deve ser gerado nenhum tipo de erro.

#### IDE (Identifier Extension Bit)

* Descrição: Identifica o tipo de *frame* de dados, Base Frame ou Extended Frame.
* Tamanho do campo: 1 bit.
* Valores esperados: Para um Extended Frame, o valor deste campo deve ser sempre recessivo (0). Caso contrário, o *frame* será um Base Frame. É interessante notar que, através dos valores codificados para os dois tipos de *frame*s de dados, o Base Frame tem maior prioridade que o Extended Frame.

#### Identifier B

* Descrição: A segunda parte do identificador único para a mensagem a ser enviada. Seu valor é utilizado na arbitração e, por isso, é responsável pela prioridade de mensagens.
* Tamanho do campo: 18 bits.
* Valores esperados: Não há restrição para os valores a serem enviados no campo de dados.

#### RTR (Remote Transmission Request)

* Descrição: Campo que identifica se uma mensagem é um *Data Frame* ou um *Remote Frame*.
* Tamanho do campo: 1 bit.
* Valores esperados: É utilizado o valor dominante (0) para um *Data Frame* e valor recessivo (1) para um *Remote Frame*.

#### Reserved Bits

* Descrição: Bits reservado para uso futuro.
* Tamanho do campo: 2 bits.
* Valores esperados: Ambos os bits devem ser dominantes (0), porém, caso um receptor receba qualquer valor diferente, não deve ser gerado nenhum tipo de erro.

#### DLC (Data Length Code)

* Descrição: Define a quantidade de dados que serão enviados no campo Data Field, de forma que o valor recebido neste campo é igual a quantidade de bytes que serão recebidos no campo Data Field.
* Tamanho do campo: 4 bits.
* Valores esperados: São esperados valores de 0 a 8. Como o tamanho do campo permite o envio de valores até 15, caso um receptor receba um valor maior que 8, deve ser considerado apenas 8 e não deve ser gerado nenhum tipo de erro.

#### Data Field

* Descrição: Dados da mensagem.
* Tamanho do campo: É definido pelo valor recebido no campo. Pode variar de 0 a 64 bits, sempre em múltiplo de 8.
* Valores esperados: Não há restrição para os valores a serem enviados no campo de dados.

#### CRC (Cyclic Redundancy Check)

* Descrição: Utilizado para identificar erros na transmissão dos dados.
* Tamanho do campo: 15 bits.
* Valores esperados: É esperado o valor resultante do cálculo de CRC. Um valor diferente deste levará a um CRC ERROR, que deve ser enviado apenas após o ACK Delimiter.

#### CRC Delimiter

* Descrição: É o bit seguinte ao CRC.
* Tamanho do campo: 1 bit.
* Valores esperados: É esperado um valor recessivo (1) para o CRC Delimiter. Um valor dominante (0) deve levar a um Form Error.

#### ACK Slot

* Descrição: Ao receber uma mensagem corretamente, um receptor deve enviar, através deste campo, sua confirmação de recebimento.
* Tamanho do campo: 1 bit.
* Valores esperados: Quando um receptor recebe uma mensagem corretamente o valor esperado no barramento é dominante (0). Quando nenhum receptor responde, ou seja, deixa o barramento em recessivo (1), o transmissor irá emitir um ACK Error.

#### ACK Delimiter

* Descrição: É o bit seguinte ao ACK Slot. Faz com que o ACK Slot seja cercado de bits recessivos.
* Tamanho do campo: 1 bit.
* Valores esperados: É esperado um valor recessivo (1) para o ACK Delimiter. Um valor dominante (0) deve levar a um Form Error.

#### EOF (End of Frame)

* Descrição: Determina o fim do *frame* de dados.
* Tamanho do campo: 7 bits.
* Valores esperados: Todos os bits devem ser recessivos (1), porém, para receptores, o último bit deve ser tratado como don’t care, ou seja, pode ser tanto dominante (0) quanto recessivo (1). Um valor diferente do esperado deve levar a um Form Error.

[https://www-inst.eecs.berkeley.edu/~ee249/fa08/Lectures/handout\_canbus2.pdf]

## Remote Frame

O *Remote Frame* possui o mesmo padrão de um *Data Frame*. O *Remote Frame* é utilizado como requisição de um determinado tipo de dado identificado pelo campo ID, de maneira que apenas o nó responsável por fornecer aquele dado é que irá responder à requisição (com um *Data Frame*). No *Remote Frame* o campo RTR é recessivo (1); o campo DLC indica o tamanho do dado que está sendo requisitado, ou seja o valor do DLC não representa o tamanho do campo de dados do *Remote Frame*, mas sim o tamanho do campo de dados da resposta (*Data Frame*) que virá daquela requisição; e o campo de dados do *Remote Frame* não é utilizado, tendo sempre comprimento de zero bits.

## Error Frame

O *Error Frame* é tipo de mensagem transmitida quando um nó detecta um erro e pode ser de dois tipos: Erro Ativo ou Erro Passivo. É composto por dois campos *Error Flags* e *Error Delimiter*.

Um ao detectar um erro deverá iniciar a transmissão do *Error Flags* a partir do bit posterior ao qual o erro foi detectado, com exceção do CRC Error que irá iniciar a transmissão do *Error Flags* após o ACK Slot.

### Superposição

Por conta de possíveis atrasos de um nó para outro na detecção de um Form Error em um *Data Frame* ou *Remote Frame*, um nó pode iniciar a transmissão do *Error Flags* depois de outro, fazendo com que haja superposição na transmissão do campo *Error Flags*. E, no pior dos casos, se um determinado nó não conseguir detectar um Form Error em um *Data Frame* ou *Remote Frame* e apenas vier a conseguir detectar um erro de *Bit Stuffing* na transmissão do *Error Flags* porque ainda estava tratando os bits recebidos como um *Data Frame* ou *Remote Frame*, então este nó (atrasado) irá iniciar a transmissão do *Error Flags* fazendo com que os outros nós que já reconheceram o *Error Flags* transmitidos anteriormente, os recebam novamente. Este é o motivo do campo *Error Flags* ter tamanho variável que vai até o dobro do seu tamanho padrão.

### Error Flags

* Descrição: Parte inicial do *Error Frame*. Possui tamanho variável por conta da superposição na transmissão do *Error Frame* por diferentes nós ao mesmo tempo com atraso no início da transmissão.
* Tamanho do campo: 6 a 12 bits. Deverá transmitir 6 bits, mas deverá suportar a recepção de até 12 bits por conta da superposição.
* Valores esperados: Todos dominantes (0), para Erro Ativo, todos recessivos (1), para Erro Passivo.

### Error Delimiter

* Descrição: Parte final do *Error Frame*.
* Tamanho do campo: 8 bits.
* Valores esperados: Todos os bits recessivos (1).

## Overload Frame

É um tipo de mensagem transmitida em três situações: 1) Quando um nó que necessita de um tempo adicional para receber o próximo *Data Frame* ou *Remote Frame*. 2) Detecção de um bit dominante (0) no primeiro ou segundo bit do *Intermission*. 3) Se um nó CAN receber um bit dominante no oitavo bit (último bit) do *Error Delimiter* ou *Overload Delimiter*, então neste último caso será transmitido um *Overload Frame* e não um *Error Frame* (como seria esperado) e o contador de erros não será alterado. O *Overload Frame* é composto pelos campos *Overload Flags* e *Overload Delimiter*.

### Overload Flags

* Descrição: Parte inicial do *Overload Frame*. Possui tamanho variável por conta da superposição, assim como no *Error Frame*.
* Tamanho do campo: 6 a 12 bits. Deverá transmitir 6 bits, mas deverá suportar a recepção de até 12 bits por conta da superposição.
* Valores esperados: Todos dominantes (0).

### Overload Delimiter

* Descrição: Parte final do *Overload Frame*.
* Tamanho do campo: 8 bits.
* Valores esperados: Todos os bits recessivos (1).

## Interframe Spacing

*Data Frame*s ou *Remote Frame*s sempre serão separados por um campo de bits chamado *Interframe Spacing* que foram precedidos de um *frame* de qualquer tipo (*Data Frame*, *Remote Frame*, *Error Frame* ou *Overload Frame*). É composto de duas partes: o *Intermission* e o Bus Idle.

### Intermission

* Descrição: Durante o *Intermission* nenhum nó poderá transmitir um *Data Frame* ou *Remote Frame*. O único *frame* que pode ser transmitido por outro nó durante a transmissão de um *Intermission* é um *Overload Frame*, então se na transmissão do primeiro ou segundo bit do *Intermission* for lido o valor dominante (0), este bit será interpretado como o primeiro bit do *Overload Flags*. E, por conta da tolerância da duração do bit, caso o terceiro bit (último bit) do *Intermission* seja lido o valor dominante (0), então este bit será reconhecido como o SOF (*Start of Frame*) do próximo *Data Frame* ou *Remote Frame*.
* Tamanho do campo: 3 bits.
* Valores esperados: Todos recessivos (1).

### Bus Idle

* Descrição: Quando nenhum *frame* está sendo transmitido, ou seja, o barramento está livre para transmissão de novos *frame*s.
* Tamanho do campo: 0 a infinitos bits.
* Valores esperados: Todos recessivos (1). Caso seja recebido um bit dominante (0), então este bit será interpretado como o SOF (*Start of Frame*) do próximo *Data Frame* ou *Remote Frame*.

# Cálculo do CRC

O cálculo do CRC é realizado sobre os bits recebidos do campo SOF (*Start of Frame*) ao Data Field, não contabilizando os bits adicionados pelo *Bit Stuffing*, o que acaba por aumentar a vulnerabilidade a erros [Di Natale]. O valor inicial utilizado é zero e o polinômio é X15 + X14 + X10 + X8 + X7 + X4 + X3 + 1. O algoritmo, conforme a especificação [BOSCH], funciona da seguinte forma:

CRC\_RG = 0; // initialize shift register

REPEAT

CRCNXT = NXTBIT EXOR CRC\_RG(14);

CRC\_RG(14:1) = CRC\_RG(13:0); // shift left by 1 position

CRC\_RG(0) = 0;

IF CRCNXT THEN

CRC\_RG(14:0) = CRC\_RG(14:0) EXOR (4599hex);

ENDIF

UNTIL (CRC SEQUENCE starts or there is an ERROR condition)

# Tratamento do *Bit Stuffing*

*Data Frame*s ou *Remote Frame*s sempre serão separados por um campo de bits chamado *Interframe Spacing* que foram precedidos de um *frame* de qualquer tipo (*Data Frame*, *Remote Frame*, *Error Frame* ou *Overload Frame*). É composto de duas partes: o *Intermission* e o Bus Idle.

# Máquina de Estados Finitos

Falar brevemente o que é uma FSM e justificar o uso

Informações importantes para o entendimento da modelagem realizada:

* rx\_bit representa o último bit recebido no estado atual.
* rx\_bit == X representa a leitura de um bit de valor qualquer.
* contador\_ESTADO representa a quantidade bits recebidos no determinado ESTADO.
* Em todos os estados em que um *Bit Stuffing* pode ocorrer, do estado S1 (ID\_A) ao S10 (CRC), o bit recebido não é armazenado se for um *Bit Stuffing*, neste caso o próximo será armazenado.
* O CRC é calculado em um componente separado. Os bits que serão utilizados no cálculo do CRC são os bits recebidos do estado S1 (ID\_A) até o estado S9 (Data).

Na modelagem da máquina de estados, os seguintes estados foram criados:

## S0 - IDLE

**Condições de entrada:**

* + Ligar o sistema (Estado inicial da máquina de estados)
  + Estado == S16 (*Intermission*) && rx\_bit == 1 && contador\_intermission == 3

**Condições de saída:**

* + rx\_bit == 0, vai para S1 (ID\_A).

**Ação:**

* + Esperar por rx\_bit == 0

## S1 - ID\_A

**Condições de entrada:**

* + Estado == S0 (Idle) && rx\_bit == 0
  + Estado == S16 (*Intermission*) && rx\_bit == 0 && contador\_intermission == 3

**Condições de saída:**

* + contador\_id\_a == 11 (Leitura de quaisquer 11 bits), vai para S2 (RTR\_SRR\_temp).
  + Erro de bit stuffing, vai para S17 (*Error Flags*).

**Ação:**

* + Contador inicia com valor 0 (contador\_id\_a = 0)
  + Habilitar verificação de bit stuffing
  + Armazenar o bit
  + Incrementar contador\_id\_a

## S2 - RTR\_SRR\_temp

**Condições de entrada:**

* + Estado == S1 (ID\_A) && contador\_ID\_A == 11

**Condições de saída:**

* + rx\_bit == X, vai para S3(IDE).
  + Erro de bit stuffing, vai para S17 (*Error Flags*).

**Ação:**

* + Armazenar o bit

## S3 - IDE

**Condições de entrada:**

* + Estado == S2 && rx\_bit == X

**Condições de saída:**

* + ide == 0, vai para S7(Bit reservado 0)
  + ide == 1, vai para S4(ID\_B)
  + Erro de bit stuffing, vai para S17 (*Error Flags*).

**Ação:**

* + Armazenar o bit

## S4 - ID\_B

**Condições de entrada:**

* + Estado == S3 (IDE) && ide == 1 (Remover da imagem a necessidade do SRR ser igual a 1)

**Condições de saída:**

* + contador\_ID\_B == 18, vai para S5 (RTR)
  + Erro de *Bit Stuffing*, vai para S17 (*Error Flags*).

**Ação:**

* + Contador inicia com valor 0 (contador\_id\_b = 0)
  + SRR = RTR\_SRR\_temp (ao entrar no estado)
  + Armazenar o bit
  + Incrementar contador\_id\_b

## S5 - RTR

**Condições de entrada:**

* + Estado == S4 (ID\_B) && contador\_ID\_B == 18

**Condições de saída:**

* + rx\_bit == X, vai para S6(Bit reservado 1).
  + Erro de bit stuffing, vai para S17 (*Error Flags*).

**Ação:**

* + Armazenar o bit

## S6 - Bit reservado 1

## **Condições de entrada:**

* + Estado == S5 (RTR) && rx\_bit == X

**Condições de saída:**

* + rx\_bit == X, vai para S7(Bit reservado 0)
  + Erro de bit stuffing, vai para S17 (*Error Flags*).

**Ação:**

* + Armazenar o bit

## S7 - Bit reservado 0

**Condições de entrada:**

Estado == S6 (Bit reservado 1) && rx\_bit == X

Estado == S3 (IDE) && ide == 0

**Condições de saída:**

rx\_bit == X, vai para S8(DLC)

Erro de bit stuffing, vai para S17 (*Error Flags*).

**Ação:**

Armazenar o bit

## S8 - DLC

**Condições de entrada:**

* + Estado == S7 (Bit reservado 0) && rx\_bit == X

**Condições de saída:**

* + contador\_dlc == 4 && rtr == 0 && dlc != 0, vai para S9 (Data)
  + contador\_dlc == 4 && (rtr == 1 || dlc ==0), vai para S10 (CRC)
  + Erro de bit stuffing, vai para S17 (*Error Flags*).

**Ação:**

* + Contador inicia com valor 0 (contador\_dlc = 0)
  + Armazenar o bit
  + Se DLC > 8, DLC = 8
  + Incrementar contador\_dlc

## S9 - Data

**Condições de entrada:**

* + Estado == S8 (DLC) && contador\_dlc == 4 && rtr == 0 && dlc != 0

**Condições de saída:**

* + contador\_data == 8 \* DLC (Adicionar condição na imagem), vai para S10 (CRC)
  + Erro de bit stuffing, vai para S17 (*Error Flags*).

**Ação:**

* + Contador inicia com valor 0 (contador\_data = 0)
  + Armazenar o bit
  + Incrementar contador\_data

## S10 - CRC

**Condições de entrada:**

* + Estado == S9 (Data) && contador\_data == 8 \* DLC
  + Estado == S8 (DLC) && contador\_dlc == 4 && (rtr == 1 || dlc ==0)

**Condições de saída:**

* + contador\_crc == 15, vai para S11 (CRC Delimiter)
  + Erro de bit stuffing, vai para S17 (*Error Flags*).

**Ação:**

* + Contador inicia com valor 0 (contador\_crc = 0)
  + Armazenar o bit
  + Incrementar contador\_crc

## S11 - CRC Delimiter

**Condições de entrada:**

* + Estado == S10 (CRC) && contador\_crc == 15

**Condições de saída:**

* + crc\_delimiter == 1, vai para S12 (ACK Slot)
  + crc\_delimiter == 0, Erro de forma, vai para S17 (*Error Flags*).

**Ação:**

* + Armazenar o bit

## S12 - ACK Slot

**Condições de entrada:**

* + Estado == S11 (CRC Delimiter) && crc\_delimiter == 1

**Condições de saída:**

* + ack\_slot == 0, vai para S13 (ACK Delimiter)
  + ack\_slot == 1, espera-se que o transmissor emita um ACKNOWLEDGMENT ERROR, vai para S17 (*Error Flags*).

**Ação:**

* + Armazenar o bit

## S13 - ACK Delimiter

**Condições de entrada:**

* + Estado == S12 (ACK Slot) && ack\_slot == 0

**Condições de saída:**

* + ack\_delimiter == 1, vai para S14 (EOF)
  + ack\_delimiter == 0 || crc != calculated\_crc, Erro de forma ou de CRC, vai para S17 (*Error Flags*).

**Ação:**

* + Armazenar o bit

## S14 - End of Frame (EOF)

**Condições de entrada:**

* + Estado == S13 (ACK Slot) && ack\_delimiter == 1

**Condições de saída:**

* + contador\_eof == 7, vai para S15 (POST EOF)
  + contador\_eof < 7 && rx\_bit ==0, Erro de forma, vai para S17 (*Error Flags*).

**Ação:**

* + Contador inicia com valor 0 (contador\_eof = 0)
  + Armazenar o bit
  + Incrementar contador\_eof

## S15 - POST EOF

**Condições de entrada:**

* + Estado == S14 (EOF) && contador\_end\_of\_*frame* == 7 && eof == b111111

**Condições de saída:**

* + rx\_bit == 1, vai para S16 (*Intermission*)
  + rx\_bit == 0, vai para S20 (*Overload Flags*)

**Ação:**

* + Armazenar o bit

## S16 - Intermission

**Condições de entrada:**

* + Estado == S15 (POST EOF) && rx\_bit == 1
  + Estado == S19 (POST *Error Delimiter*) && rx\_bit ==1
  + Estado == S22 (POST *Overload Delimiter*) && rx\_bit ==1

**Condições de saída:**

* + contador\_intermission < 3 && rx\_bit == 0, vai para S20 (*Overload Flags*)
  + contador\_intermission == 3 && rx\_bit ==0, vai pra S1 (ID\_A)
  + contador\_intermission == 3 && rx\_bit ==1, vai pra S0 (IDLE)

**Ação:**

* + Contador inicia com valor 1 (contador\_intermission = 1)
  + Armazenar o bit
  + Incrementar contador\_intermission

## S17 - Error Flags

**Condições de entrada:**

* + Estado == S1 (ID\_A) && Erro de bit Stuffing
  + Estado == S2 (RTR\_SRR) && Erro de bit Stuffing
  + Estado == S3 (IDE) && Erro de bit Stuffing
  + Estado == S4 (ID\_B) && Erro de bit Stuffing
  + Estado == S5 (RTR) && Erro de bit Stuffing
  + Estado == S6 (Bit Reservado 1) && Erro de bit Stuffing
  + Estado == S7 (Bit Reservado 0) && Erro de bit Stuffing
  + Estado == S8 (DLC) && Erro de bit Stuffing
  + Estado == S9 (Data) && Erro de bit Stuffing
  + Estado == S10 (CRC) && Erro de bit Stuffing
  + Estado == S11 (CRC Delimiter) && crc\_delimiter == 0 (Erro de forma)
  + Estado == S12 (ACK Slot) && ack\_slot == 1 (Erro de ACK)
  + Estado == S13 (ACK Delimiter) && ack\_delimiter == 1 (Erro de forma)
  + Estado == S13 (ACK Delimiter) && crc != calculated\_crc (Erro de CRC)
  + Estado == S14 (EOF) && contador\_eof < 7 && rx\_bit ==0
  + Estado == S20 (*Overload Flags*) && contador\_flags > 12
  + Estado == S20 (*Overload Flags*) && contador\_flags < 6 && rx\_bit == 1
  + Estado == S21 (*Overload Delimiter*) && contador\_delimiter < 8 && rx\_bit == 0
  + Estado == S18 (*Error Delimiter*) && contador\_delimiter < 8 && rx\_bit == 0

**Condições de saída:**

* + contador\_flags >= 6 && contador\_flags <12 && rx\_bit == 1, vai para S18 (*Error Delimiter*)

**Ação:**

* + Contador inicia com valor 0 (contador\_flags = 0)
  + Armazenar o bit
  + Incrementar contador\_flags
  + Zera o contador\_flags caso detecte um bit de valor 1 quando o contador estiver menor que 6
  + Zera o contador\_flags caso o contador tenha valor maior que 12 (FIXME - Isso é correto?)

## S18 - Error Delimiter

**Condições de entrada:**

* + Estado == S17 (*Error Flags*) && contador\_flags >= 6 && contador\_flags < 12 && rx\_bit == 1

**Condições de saída:**

* + contador\_delimiter == 8 && delimiter == b1111111, vai para S19 (POST *Error Delimiter*)
  + contador\_delimiter < 8 && rx\_bit == 0, vai para S17 (*Error Flags*)

**Ação:**

* + Contador inicia com valor 0 (contador\_delimiter = 0)
  + Armazenar o bit
  + Incrementar contador\_delimiter

## S19 - POST Error Delimiter

**Condições de entrada:**

* + Estado == S18 (*Error Delimiter*) && contador\_delimiter == 8 && delimiter == b1111111

**Condições de saída:**

* + rx\_bit == 1, vai para S16 (*Intermission*)
  + rx\_bit == 0, vai para S20 (*Overload Flags*)

**Ação:**

* + Armazenar o bit

## S20 - Overload Flags

**Condições de entrada:**

* + Estado == S15 (POST EOF) && rx\_bit == 0
  + Estado == S16 (*Intermission*) && contador\_intermission < 3 && rx\_bit == 0
  + Estado == S19 (POST *Error Delimiter*) && rx\_bit == 0
  + Estado == S22 (Last Bit *Overload Delimiter*) && rx\_bit == 0

**Condições de saída:**

* + contador\_flags >= 6 && contador\_flags < 12 && rx\_bit == 1, vai para S21 (*Overload Delimiter*)
  + contador\_flags > 12, vai para S17 (*Error Flags*)
  + contador\_flags < 6 && rx\_bit == 1, vai para S17 (*Error Flags*)

**Ação:**

* + Para as condições de entrada 1,3 e 4 : contador inicia com valor 1 (contador\_flags = 1)
  + Para a condição de entrada 2 : contador inicia com valor 0 (contador\_flags = 0)
  + Armazenar o bit
  + Incrementar contador\_flags

## S21 - Overload Delimiter

**Condições de entrada:**

* + Estado == S20 (*Overload Flags*) && contador\_flags >= 6 && contador\_flags < 12 && rx\_bit == 1

**Condições de saída:**

* + contador\_delimiter == 8 && delimiter == b1111111, vai para S22 (POST *Overload Delimiter*)
  + contador\_delimiter < 8 && rx\_bit == 0, vai para S17 (*Error Flags*)

**Ação:**

* + Armazenar o bit
  + Incrementar contador\_delimiter

## S22 - POST Overload Delimiter

**Condições de entrada:**

* + Estado == S21 (*Overload Delimiter*) && contador\_delimiter == 8 && delimiter == b1111111

**Condições de saída:**

* + rx\_bit == 1, vai para S16 (*Intermission*)
  + rx\_bit == 0, vai para S20 (*Overload Flags*)

**Ação:**

* + Armazenar o bit

# Implementação em Verilog de um Decodificador de *Frames* CAN 2.0

A implementação foi dividida em três módulos:

* + tester.v - Responsável por gerar os sinais de entrada para teste e o clock.
  + can\_decoder.v - Contém toda a implementação do decoder, incluindo a máquina de estados e o tratamento do bit stuffing.
  + can\_crc.v - Responsável pelo cálculo do CRC.

## Módulo CAN Decoder (can\_decoder.v)

### Máquina de estados

Para implementar a máquina de estados foram utilizados dois blocos always para cada estado, sendo um responsável pelo gerenciamento do estado e outro pelo armazenamento dos dados lidos.

Para exemplificação da lógica utilizada na implementação dos estados, as subseções a seguir demonstram a implementação do Estado DLC.

#### https://lh4.googleusercontent.com/qx-wf6lTgbPKSg_9yHOgGm-ldbgqrqS-xQBmhJo8rcx57ZLTq9Gv6m9ABk7foBssyWVcdayxfaAcNe6qjIF4NQSSTatlinDUtdzj5_-B_L2jLpwG2tTq_en30q4ZGDkKFDyXmorXGerenciamento do estado

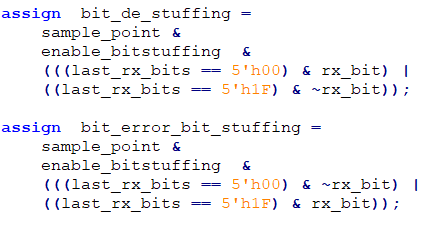
Fig. 1 - Código em Verilog da lógica sequencial para tratamento do estado DLC.

Pelo código mostrado na Fig. 1, pode-se interpretar o comportamento do estado para cada condição presente no código:

* + Após um reset, o estado não será o DLC.
  + Se forem atingidas as condições para ir para os estados DATA, CRC ou ERROR FLAGS, que são os estados para os quais existe uma transição partindo de DLC, a variável state\_dlc terá o valor 0, indicando que o estado atual não é o DLC.
  + Se as condições para ir para o estado DLC forem atingidas (o valor de go\_state\_dlc é 1), então a variável state\_dlc terá valor 1, indicando que o estado atual é o DLC.

#### https://lh4.googleusercontent.com/49g0C5XT1vlmCsrQadvkryFxtSYEONiAkUOvjchHOu69Vidf5sC4Cg8kK3hVgq_k7qKXN4WkBN8dSYfz4EqdbueUzyO2o9Rn6KfBskUq1ee2krtB0Fr8mogRq-nxC42kccavqkcDArmazenamento dos dados lidos no estado

Fig. 2 - Código em Verilog da lógica sequencial para tratamento dos bits lidos no estado DLC.

Pelo código mostrado na Fig. 2 acima, pode-se interpretar o comportamento do estado para cada condição presente no código:

* + Após um reset, o contador utilizado no estado DLC e o valor armazenado no campo DLC serão zerados.
  + Se houve a leitura de um bit que não deve ser ignorado (*Bit Stuffing*) no estado DLC, então o bit é armazenado no campo DLC e o contador DLC é incrementado.
  + Se o valor do campo DLC for maior que 8, então é alterado para 8.
  + Se houve a leitura de um bit em um estado que não é o DLC, o contador do DLC é zerado, assim começando com valor zero quando entrar novamente no estado DLC.

Fig. 3 - Código em Verilog da lógica combinacional de checagem do bit stuffing.

### Tratamento do Bit Stuffing

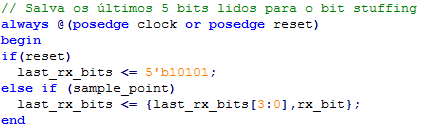
O tratamento do *Bit Stuffing* foi implementado utilizando um bloco always e 2 assigns. O bloco always tem como função armazenar os últimos 5 bits lidos, enquanto os assigns detectam quando existe um erro de *Bit Stuffing* ou quando o bit lido deverá ser ignorado.

Fig. 4 - Código em Verilog da lógica sequencial para tratamento do bit stuffing.

Pelo código mostrado na Fig. 4, pode-se interpretar o comportamento do código:

* + Após um reset, a variável contendo os últimos 5 bits lidos é atribuída a um valor qualquer, que não interfere na corretude da funcionalidade.
  + Se houve a leitura de um bit, será realizado um *shift left* na variável contendo os últimos 5 bits e o bit menos significativo será atribuído ao valor do bit lido.

Pelo código mostrado na Fig. 3, pode-se interpretar o comportamento do código:

* + Para detectar que o bit lido deve ser ignorado, os últimos 5 bits lidos devem ser 0’s e o bit lido atual deve ser 1; OU os últimos bits lidos devem ser 1’s e o bit lido atual deve ser 0.
  + Para detectar um erro de *Bit Stuffing*, os últimos 5 bits lidos devem ser 0’s e o bit lido atual deve ser 0; OU os últimos 5 bits lidos devem ser 1’s e o bit lido atual deve ser 1.

## Módulo CAN CRC (can\_crc.v)

### Cálculo do CRC

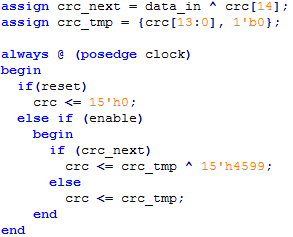
Para o cálculo do CRC foi implementado em verilog exatamente o mesmo algoritmo presente na especificação da CAN.

Fig. 5 - Código em Verilog da Lógica sequencial para o cálculo do CRC.

# Casos de teste

## Casos simples

(1)

### Base Data Frame

Tabela I - Base Data Frame

|  |  |
| --- | --- |
| Campo | Valor em binário (valor em hexadecimal) |
| SOF | 0 |
| ID | 11001110010 (0x672) |
| RTR | 0 |
| IDE | 0 |
| Reserved | 0 |
| DLC | 1000 (0x8) |
| DATA | 1010101010101010101010101010101010101010101010101010101010101010  (0xAAAAAAAAAAAAAAAA) |
| CRC | 000000001010001 (0x51) |
| CRC Delimiter | 1 |
| ACK Slot | 0 |
| ACK Delimiter | 1 |
| EOF | 1111111 |
|  |  |
| Frame completo com ***Bit Stuffing*** | 011001110010000100010101010101010101010101010101010101010101010101010101010101010100000**1**000010100011011111111 |

(10)

### Base Remote Frame

|  |  |
| --- | --- |
| Campo | Valor em binário (valor em hexadecimal) |
| SOF | 0 |
| ID | 11001110010 (0x672) |
| RTR | 1 |
| IDE | 0 |
| Reserved | 0 |
| DLC | 0000 (0x0) |
| CRC | 100000100010000 (0x4110) |
| CRC Delimiter | 1 |
| ACK Slot | 0 |
| ACK Delimiter | 1 |
| EOF | 1111111 |
|  |  |
| Frame completo com ***Bit Stuffing*** | 011001110010100000**1**0100000**1**1000100001011111111 |

(12)

### Extended Data Frame

|  |  |
| --- | --- |
| Campo | Valor em binário (valor em hexadecimal) |
| SOF | 0 |
| ID\_A | 10001001001 (0x449) |
| SRR | 1 |
| IDE | 1 |
| ID\_B | 110000000001111010 (0x3007A) |
| RTR | 0 |
| Reserved1 | 0 |
| Reserved0 | 0 |
| DLC | 1000 (0x8) |
| DATA | 1010101010101010101010101010101010101010101010101010101010101010 (0xAAAAAAAAAAAAAAAA) |
| CRC | 111101111110101 (0x7BF5) |
| CRC Delimiter | 1 |
| ACK Slot | 0 |
| ACK Delimiter | 1 |
| EOF | 1111111 |
|  |  |
| Frame completo com ***Bit Stuffing*** | 0100010010011111**0**0000**1**00000**1**1111**0**010000100010101010101010101010101010101010101010101010101010101010101010101111011111**0**101011011111111 |

(13)

### Extended Remote Frame

|  |  |
| --- | --- |
| Campo | Valor em binário (valor em hexadecimal) |
| SOF | 0 |
| ID\_A | 10001001001 (0x449) |
| SRR | 1 |
| IDE | 1 |
| ID\_B | 110000000001111010 (0x3007A) |
| RTR | 1 |
| Reserved1 | 0 |
| Reserved0 | 0 |
| DLC | 1000 (0x8) |
| CRC | 010100111110110 (0x29F6) |
| CRC Delimiter | 1 |
| ACK Slot | 0 |
| ACK Delimiter | 1 |
| EOF | 1111111 |
|  |  |
| Frame completo com ***Bit Stuffing*** | 0100010010011111**0**0000**1**00000**1**1111**0**010100100001010011111**0**01101011111111 |

### Overload Frame

O caso de teste para o *Overload Frame* será em uma situação que pode acontecer na prática. Será enviado um *Data Frame* igual ao utilizado no caso de teste mostrado na Tabela I seguido de um *Overload Frame*, um *Intermission* e o mesmo *frame* enviado inicialmente. O *Overload Frame* enviado contém seis bits dominantes no campo *Overload Flags* e oito bits recessivos no campo *Overload Delimiter*. O *intermission* enviado contém 3 bits recessivos.

|  |  |
| --- | --- |
| Campo | Valor em binário (valor em hexadecimal) |
| Data Frame | Vide Tabela I. |
| Overload Flags | 000000 |
| Overload Delimiter | 11111111 |
| Intermission | 111 |
| Data Frame | Vide Tabela I. |

### Error Frame

Foram utilizados 3 casos de teste básicos para *Error Frame*. O primeiro simula um Erro de bit Stuffing, ou seja, contém 6 bits iguais e consecutivos. O segundo caso contém um erro no CRC, ou seja, o valor do CRC é diferente do valor esperado. O terceiro caso será enviado um bit recessivo no campo de *Ack Slot*, caracterizando um *ACK Error*. Em todos os casos, o nosso decoder, mesmo sem ter a capacidade de enviar um *frame* de erro no barramento, espera que algum nó envie o erro.

Caso 1 – Erro de Bit Stuffing

|  |  |
| --- | --- |
| Campo | Valor em binário (valor em hexadecimal) |
| SOF | 0 |
| ID | 11001110010 (0x672) |
| RTR | 0 |
| IDE | 0 |
| Reserved | 0 |
| DLC | 1000 (0x8) |
| DATA | 1010101010101010101010101010101010101010101010101010101010101010  (0xAAAAAAAAAAAAAAAA) |
| CRC Esperado | 000000001010001 (0x51) |
| Dados Recebidos no campo de CRC | 000000 – Detectado Erro de Bit Stuffing |
| Error Flags | 000000 |
| Error Delimiter | 11111111 |
|  |  |
| Frame completo com Erro de ***Bit Stuffing*** | 011001110010000100010101010101010101010101010101010101010101010101010101010101010100000000000011111111 |

Caso 2 – Erro de CRC

|  |  |
| --- | --- |
| Campo | Valor em binário (valor em hexadecimal) |
| SOF | 0 |
| ID | 11001110010 (0x672) |
| RTR | 0 |
| IDE | 0 |
| Reserved | 0 |
| DLC | 1000 (0x8) |
| DATA | 1010101010101010101010101010101010101010101010101010101010101010  (0xAAAAAAAAAAAAAAAA) |
| CRC Esperado | 000000001010001 (0x51) |
| CRC Recebido | 000000001110001 (0x71) |
| CRC Delimiter | 1 |
| ACK Slot | 0 |
| ACK Delimiter | 1 |
| Error Flags | 000000 |
| Error Delimiter | 11111111 |
|  |  |
| Frame completo com ***Bit Stuffing*** e Erro de CRC | 011001110010000100010101010101010101010101010101010101010101010101010101010101010100000**1**0000111000110100000011111111 |

Caso 3 – Erro de ACK

|  |  |
| --- | --- |
| Campo | Valor em binário (valor em hexadecimal) |
| SOF | 0 |
| ID | 11001110010 (0x672) |
| RTR | 0 |
| IDE | 0 |
| Reserved | 0 |
| DLC | 1000 (0x8) |
| DATA | 1010101010101010101010101010101010101010101010101010101010101010  (0xAAAAAAAAAAAAAAAA) |
| CRC | 000000001010001 (0x51) |
| CRC Delimiter | 1 |
| ACK Slot | 1 |
| Error Flags | 000000 |
| Error Delimiter | 11111111 |
|  |  |
| Frame completo com ***Bit Stuffing*** e Erro de ACK | 011001110010000100010101010101010101010101010101010101010101010101010101010101010100000**1**000010100011100000011111111 |

(17) = (1) + Erro de ACK + Frame de Erro

(18) = (1) + Erro de bit stuffing + Frame de Erro

(19) = (1) + Erro de CRC + Frame de Erro

## Casos complexos

### Sequência 1 (33)

A sequência 1 é a concatenação dos dados de 4 frames iguais ao mostrado na Tabela I. O principal objetivo deste caso de teste foi avaliar a capacidade do decoder receber corretamente uma sequência de frames.

(1) + (1) + (1) + (1)

### Sequência 2 (30)

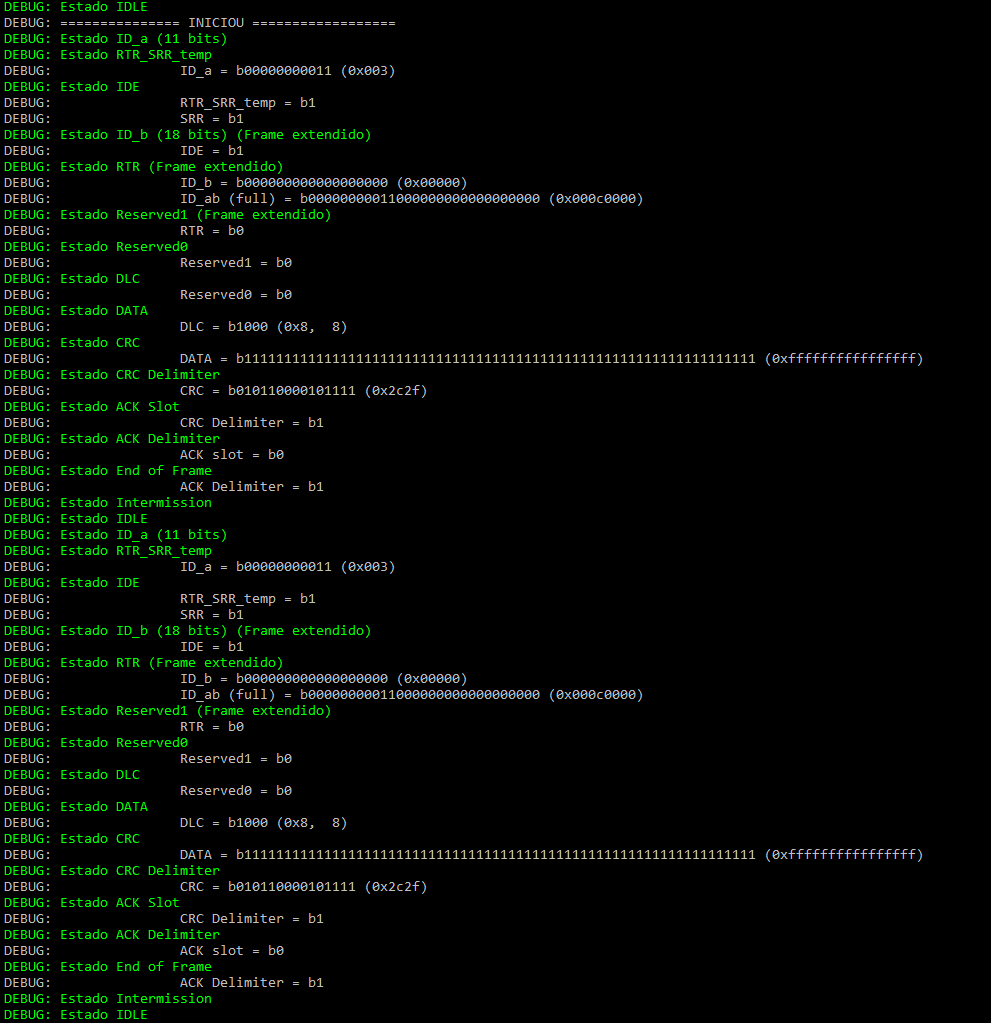
A sequência 2 possui dois frames de dados estendidos, sendo cada um destes precedido por 78 bits recessivos. Os dois frames enviados são iguais e contêm várias sequências de bits de mesmo nível lógico nos campos de ID e Data, assim sendo também um bom caso de teste para o tratamento correto do Bit Stuffing. Neste caso de teste o campo DLC é enviado com valor 15, assim testando o caso em que DLC > 8, no qual o comportamento esperado é considerar o valor 8. Além disso, o CRC do frame utilizado termina com 4 bits 1’s, fazendo com que o bit referente ao CRC delimiter, que também deve ter valor 1, seja o quinto bit de mesmo nível lógico seguido, porém o comportamento desejado é que não seja inserido um bit referente ao Bit Stuffing, pois o campo CRC Delimiter não deve sofrer a influência desta técnica.

//TODO – Adicionar tabela com os valores do frame descrito

Frame utilizado na sequência descrita acima

|  |  |
| --- | --- |
| Campo | Valor em binário (valor em hexadecimal) |
| SOF | 0 |
| ID\_A | 00000000011 (0x3) |
| SRR | 1 |
| IDE | 1 |
| ID\_B | 000000000000000000 (0x0) |
| RTR | 0 |
| Reserved1 | 0 |
| Reserved0 | 0 |
| DLC | 1111 (0xF) |
| DATA | 1111111111111111111111111111111111111111111111111111111111111111 (0xFFFFFFFFFFFFFFF) |
| CRC | 010110000101111 (0x2C2F) |
| CRC Delimiter | 1 |
| ACK Slot | 0 |
| ACK Delimiter | 1 |
| EOF | 1111111 |
|  |  |
| Frame completo com ***Bit Stuffing*** | 00000**1**00000**1**1111**0**0000**1**00000**1**00000**1**00000**1**0011111**0**11111**0**11111**0**11111**0**11111**0**11111**0**11111**0**11111**0**11111**0**11111**0**11111**0**11111**0**11111**0**1110101100001011111011111111 |

|  |  |
| --- | --- |
| Campo | Valor em binário (valor em hexadecimal) |
| IDLE | 78 x 1b |
| Frame | Vide tabela anterior. |
| Intermission | 111 |
| IDLE | 75 x 1b |
| Frame | Vide tabela anterior. |
|  |  |
| Caso de teste completo com ***Bit Stuffing*** | 11111111111111111111111111111111111111111111111111111111111111111111111111111100000**1**00000**1**1111**0**0000**1**00000**1**00000**1**00000**1**0011111**0**11111**0**11111**0**11111**0**11111**0**11111**0**11111**0**11111**0**11111**0**11111**0**11111**0**11111**0**11111**0**111010110000101111101111111111111111111111111111111111111111111111111111111111111111111111111111111111111100000**1**00000**1**1111**0**0000**1**00000**1**00000**1**00000**1**0011111**0**11111**0**11111**0**11111**0**11111**0**11111**0**11111**0**11111**0**11111**0**11111**0**11111**0**11111**0**11111**0**1110101100001011111011111111 |



### Sequência 3 (teste final – inicio do arquivo)

A sequência 3 é uma sequência longa contendo 8 frames de dados, 1 frame remoto, e 2 frames de erro. O objetivo desta sequência de teste foi avaliar a capacidade do decoder de receber uma grande quantidade de bits e continuar funcionando, mesmo após alguns frames de erro. O primeiro frame da sequência é o mesmo representado na [tabela 1]. Os três frames seguintes são semelhantes, porém com campo de dados de tamanhos 5,3 e 0. O quinto frame é o frame descrito na seção Extended Data Frame. Em seguida é adicionada a sequência de bits utilizada na seção Error Frame -CRC Error. Continuando a sequencia, é adicionado um frame igual ao representado na seção Error Frame - ACK Error, seguido de um frame semelhante ao da seção Remote Extended Frame, porém com DLC = 0 e o CRC modificado devido a esta mudança. Por fim é adicionado um frame semelhante ao da seção Base Data Frame, porém com DLC = 6. É importante notar que foram adicionados os bits referentes ao Intermission sempre que necessário.

|  |  |
| --- | --- |
| Campo | Valor em binário com **Bit Stuffing** (valor em hexadecimal) |
| Frame 1 | 011001110010000100010101010101010101010101010101010101010101010101010101010101010100000**1**000010100011011111111 |
| Intermission | 111 |
| Frame 2 | 0110011100100000**1**10110101010101010101010101010101010101010101100111001001101011111111 |
| Intermission | 111 |
| Frame 3 | 0110011100100000**1**01110101010101010101010101001001011100000**1**11011111111 |
| Intermission | 111 |
| Frame 4 | 0110011100100000**1**0000110010110101011011111111 |
| Intermission | 111 |
| Frame 5 | 0100010010011111**0**0000**1**00000**1**1111**0**010000100010101010101010101010101010101010101010101010101010101010101010101111011111**0**101011011111111 |
| Intermission | 111 |
| Frame 6 (Erro de CRC) | 011001110010000100010101010101010101010101010101010101010101010101010101010101010100000**1**00001110001101 |
| Error Frame | 00000011111111 |
| Intermission | 111 |
| Frame 7 (ACK Error) | 011001110010000100010101010101010101010101010101010101010101010101010101010101010100000**1**000010100011100000011111111 |
| Intermission | 111 |
| Frame 8 | 0100010010011111**0**0000**1**00000**1**1111**0**010100000**1**010111011111000011011111111 |
| Intermission | 111 |
| Frame 9 | 011001110010000011101010101010101010101010101010101010101010101010100001001001011101011111111 |
|  |  |
| Caso de teste completo | 011001110010000100010101010101010101010101010101010101010101010101010101010101010100000**1**0000101000110111111111110110011100100000**1**101101010101010101010101010101010101010101011001110010011010111111111110110011100100000**1**01110101010101010101010101001001011100000**1**110111111111110110011100100000**1**00001100101101010110111111111110100010010011111**0**0000**1**00000**1**1111**0**010000100010101010101010101010101010101010101010101010101010101010101010101111011111**0**101011011111111111011001110010000100010101010101010101010101010101010101010101010101010101010101010100000**1**0000111000110100000011111111111011001110010000100010101010101010101010101010101010101010101010101010101010101010100000**1**0000101000111000000111111111110100010010011111**0**0000**1**00000**1**1111**0**010100000**1**010111011111000011011111111111011001110010000011101010101010101010101010101010101010101010101010100001001001011101011111111 |

# Conclusão

Preencher aqui

# Guidelines For Manuscript Preparation

When you open trans\_jour.docx, select “Page Layout” from the “View” menu in the menu bar (View | Page Layout), (these instructions assume MS 6.0. Some versions may have alternate ways to access the same functionalities noted here). Then, type over sections of trans\_jour.docx or cut and paste from another document and use markup styles. The pull-down style menu is at the left of the Formatting Toolbar at the top of your *Word* window (for example, the style at this point in the document is “Text”). Highlight a section that you want to designate with a certain style, and then select the appropriate name on the style menu. The style will adjust your fonts and line spacing. Do not change the font sizes or line spacing to squeeze more text into a limited number of pages.Use italics for emphasis; do not underline.

## Copyright Form

Authors must submit an electronic IEEE Copyright Form (eCF) upon submitting their final manuscript files.  You can access the eCF system through your manuscript submission system or through the Author Gateway. You are responsible for obtaining any necessary approvals and/or security clearances. For additional information on intellectual property rights, visit the IEEE Intellectual Property Rights department web page at

<http://www.ieee.org/publications_standards/publications/rights/index.html>.

# IEEE Publishing Policy

The general IEEE policy requires that authors should only submit original work that has neither appeared elsewhere for publication, nor is under review for another refereed publication. The submitting author must disclose all prior publication(s) and current submissions when submitting a manuscript. Do not publish “preliminary” data or results. The submitting author is responsible for obtaining agreement of all coauthors and any consent required from employers or sponsors before submitting an article. The IEEE Transactions and Journals Department strongly discourages courtesy authorship; it is the obligation of the authors to cite only relevant prior work.

The IEEE Transactions and Journals Department does not publish conference records or proceedings, but can publish articles related to conferences that have undergone rigorous peer review. Minimally, two reviews are required for every article submitted for peer review.

# Publication Principles

The two types of contents of that are published are; 1) peer-reviewed and 2) archival. The Transactions and Journals Department publishes scholarly articles of archival value as well as tutorial expositions and critical reviews of classical subjects and topics of current interest.

Authors should consider the following points:

1. Technical papers submitted for publication must advance the state of knowledge and must cite relevant prior work.
2. The length of a submitted paper should be commensurate with the importance, or appropriate to the complexity, of the work. For example, an obvious extension of previously published work might not be appropriate for publication or might be adequately treated in just a few pages.
3. Authors must convince both peer reviewers and the editors of the scientific and technical merit of a paper; the standards of proof are higher when extraordinary or unexpected results are reported.
4. Because replication is required for scientific progress, papers submitted for publication must provide sufficient information to allow readers to perform similar experiments or calculations and use the reported results. Although not everything need be disclosed, a paper must contain new, useable, and fully described information. For example, a specimen’s chemical composition need not be reported if the main purpose of a paper is to introduce a new measurement technique. Authors should expect to be challenged by reviewers if the results are not supported by adequate data and critical details.
5. Papers that describe ongoing work or announce the latest technical achievement, which are suitable for presentation at a professional conference, may not be appropriate for publication.

References

*Basic format for books:*

J. K. Author, “Title of chapter in the book,” in *Title of His Published Book, x*th ed. City of Publisher, (only U.S. State), Country: Abbrev. of Publisher, year, ch. *x*, sec. *x*, pp. *xxx–xxx.*

*Examples:*

1. G. O. Young, “Synthetic structure of industrial plastics,” in *Plastics,* 2nd ed., vol. 3, J. Peters, Ed. New York, NY, USA: McGraw-Hill, 1964, pp. 15–64.
2. W.-K. Chen, *Linear Networks and Systems.* Belmont, CA, USA: Wadsworth, 1993, pp. 123–135.

*Basic format for periodicals:*

J. K. Author, “Name of paper,” *Abbrev. Title of Periodical*, vol. *x, no*. *x,* pp*. xxx-xxx,* Abbrev. Month, year, DOI. 10.1109.*XXX*.123456.

*Examples:*

1. J. U. Duncombe, “Infrared navigation—Part I: An assessment of feasibility,” *IEEE Trans. Electron Devices*, vol. ED-11, no. 1, pp. 34–39, Jan. 1959, 10.1109/TED.2016.2628402.
2. E. P. Wigner, “Theory of traveling-wave optical laser,”   
   *Phys. Rev*.,   
   vol. 134, pp. A635–A646, Dec. 1965.
3. E. H. Miller, “A note on reflector arrays,” *IEEE Trans. Antennas Propagat*., to be published.

*Basic format for reports:*

J. K. Author, “Title of report,” Abbrev. Name of Co., City of Co., Abbrev. State, Country, Rep. *xxx*, year.

*Examples:*

1. E. E. Reber, R. L. Michell, and C. J. Carter, “Oxygen absorption in the earth’s atmosphere,” Aerospace Corp., Los Angeles, CA, USA, Tech. Rep. TR-0200 (4230-46)-3, Nov. 1988.
2. J. H. Davis and J. R. Cogdell, “Calibration program for the 16-foot antenna,” Elect. Eng. Res. Lab., Univ. Texas, Austin, TX, USA, Tech. Memo. NGL-006-69-3, Nov. 15, 1987.

*Basic format for handbooks:*

*Name of Manual/Handbook, x* ed., Abbrev. Name of Co., City of Co., Abbrev. State, Country, year, pp. *xxx-xxx.*

*Examples:*

1. *Transmission Systems for Communications*, 3rd ed., Western Electric Co., Winston-Salem, NC, USA, 1985, pp. 44–60.
2. *Motorola Semiconductor Data Manual*, Motorola Semiconductor Products Inc., Phoenix, AZ, USA, 1989.

*Basic format for books (when available online):*

J. K. Author, “Title of chapter in the book,” in *Title of Published Book*, *x*th ed. City of Publisher, State, Country: Abbrev. of Publisher, year, ch. *x*, sec. *x*, pp. *xxx–xxx*. [Online]. Available: http://www.web.com

*Examples:*

1. G. O. Young, “Synthetic structure of industrial plastics,” in Plastics, vol. 3, Polymers of Hexadromicon, J. Peters, Ed., 2nd ed. New York, NY, USA: McGraw-Hill, 1964, pp. 15-64. [Online]. Available: http://www.bookref.com.
2. *The Founders’ Constitution*, Philip B. Kurland and Ralph Lerner, eds., Chicago, IL, USA: Univ. Chicago Press, 1987. [Online]. Available: http://press-pubs.uchicago.edu/founders/
3. The Terahertz Wave eBook. ZOmega Terahertz Corp., 2014. [Online]. Available: http://dl.z-thz.com/eBook/zomega\_ebook\_pdf\_1206\_sr.pdf. Accessed on: May 19, 2014.
4. Philip B. Kurland and Ralph Lerner, eds., *The Founders’ Constitution.* Chicago, IL, USA: Univ. of Chicago Press, 1987, Accessed on: Feb. 28, 2010, [Online] Available: http://press-pubs.uchicago.edu/founders/

*Basic format for journals (when available online):*

J. K. Author, “Name of paper,” *Abbrev. Title of Periodical*, vol. *x*, no. *x*, pp. *xxx-xxx*, Abbrev. Month, year. Accessed on: Month, Day, year, DOI: 10.1109.*XXX*.123456, [Online].

*Examples:*

1. J. S. Turner, “New directions in communications,” *IEEE J. Sel. Areas Commun*., vol. 13, no. 1, pp. 11-23, Jan. 1995.
2. W. P. Risk, G. S. Kino, and H. J. Shaw, “Fiber-optic frequency shifter using a surface acoustic wave incident at an oblique angle,” *Opt. Lett.*, vol. 11, no. 2, pp. 115–117, Feb. 1986.
3. P. Kopyt *et al., “*Electric properties of graphene-based conductive layers from DC up to terahertz range,” *IEEE THz Sci. Technol.,* to be published. DOI: 10.1109/TTHZ.2016.2544142.

*Basic format for papers presented at conferences (when available online):*

J.K. Author. (year, month). Title. presented at abbrev. conference title. [Type of Medium]. Available: site/path/file

*Example:*

1. PROCESS Corporation, Boston, MA, USA. Intranets: Internet technologies deployed behind the firewall for corporate productivity. Presented at INET96 Annual Meeting. [Online]. Available: http://home.process.com/Intranets/wp2.htp

*Basic format for reports and handbooks (when available online):*

J. K. Author. “Title of report,” Company. City, State, Country. Rep. no., (optional: vol./issue), Date. [Online] Available: site/path/file

*Examples:*

1. R. J. Hijmans and J. van Etten, “Raster: Geographic analysis and modeling with raster data,” R Package Version 2.0-12, Jan. 12, 2012. [Online]. Available: http://CRAN.R-project.org/package=raster
2. Teralyzer. Lytera UG, Kirchhain, Germany [Online]. Available: http://www.lytera.de/Terahertz\_THz\_Spectroscopy.php?id=home, Accessed on: Jun. 5, 2014

*Basic format for computer programs and electronic documents (when available online):*

Legislative body. Number of Congress, Session. (year, month day). *Number of bill or resolution*, *Title*. [Type of medium]. Available: site/path/file

***NOTE:*** ISO recommends that capitalization follow the accepted practice for the language or script in which the information is given.

*Example:*

1. U.S. House. 102nd Congress, 1st Session. (1991, Jan. 11). *H. Con. Res. 1, Sense of the Congress on Approval of Military Action*. [Online]. Available: LEXIS Library: GENFED File: BILLS

*Basic format for patents (when available online):*

Name of the invention, by inventor’s name. (year, month day). Patent Number[Type of medium]. Available: site/path/file

*Example:*

1. Musical toothbrush with mirror, by L.M.R. Brooks. (1992, May 19). Patent D 326 189

[Online]. Available: NEXIS Library: LEXPAT File: DES

*Basic format for conference proceedings (published):*

J. K. Author, “Title of paper,” in *Abbreviated Name of Conf.*, City of Conf., Abbrev. State (if given), Country, year, pp. *xxxxxx.*

*Example:*

1. D. B. Payne and J. R. Stern, “Wavelength-switched pas- sively coupled single-mode optical network,” in *Proc. IOOC-ECOC,* Boston, MA, USA,1985,   
   pp. 585–590.

*Example for papers presented at conferences (unpublished):*

1. D. Ebehard and E. Voges, “Digital single sideband detection for interferometric sensors,” presented at the *2nd Int. Conf. Optical Fiber Sensors,* Stuttgart, Germany, Jan. 2-5, 1984.

*Basic format for patents:*

J. K. Author, “Title of patent,” U.S. Patent *x xxx xxx*, Abbrev. Month, day, year.

*Example:*

1. G. Brandli and M. Dick, “Alternating current fed power supply,” U.S. Patent 4 084 217, Nov. 4, 1978.

*Basic format**for theses (M.S.) and dissertations (Ph.D.):*

a) J. K. Author, “Title of thesis,” M.S. thesis, Abbrev. Dept., Abbrev. Univ., City of Univ., Abbrev. State, year.

b) J. K. Author, “Title of dissertation,” Ph.D. dissertation, Abbrev. Dept., Abbrev. Univ., City of Univ., Abbrev. State, year.

*Examples:*

1. J. O. Williams, “Narrow-band analyzer,” Ph.D. dissertation, Dept. Elect. Eng., Harvard Univ., Cambridge, MA, USA, 1993.
2. N. Kawasaki, “Parametric study of thermal and chemical nonequilibrium nozzle flow,” M.S. thesis, Dept. Electron. Eng., Osaka Univ., Osaka, Japan, 1993.

*Basic format for the most common types of unpublished references:*

a) J. K. Author, private communication, Abbrev. Month, year.

b) J. K. Author, “Title of paper,” unpublished.

c) J. K. Author, “Title of paper,” to be published.

*Examples:*

1. A. Harrison, private communication, May 1995.
2. B. Smith, “An approach to graphs of linear forms,” unpublished.
3. A. Brahms, “Representation error for real numbers in binary computer arithmetic,” IEEE Computer Group Repository, Paper R-67-85.

*Basic formats for standards:*

a) *Title of Standard*, Standard number, date.

b) *Title of Standard*, Standard number, Corporate author, location, date.

*Examples:*

1. IEEE Criteria for Class IE Electric Systems, IEEE Standard 308, 1969.
2. Letter Symbols for Quantities, ANSI Standard Y10.5-1968.

*Article number in reference examples:*

1. R. Fardel, M. Nagel, F. Nuesch, T. Lippert, and A. Wokaun, “Fabrication of organic light emitting diode pixels by laser-assisted forward transfer,” *Appl. Phys. Lett.*, vol. 91, no. 6, Aug. 2007, Art. no. 061103.
2. J. Zhang and N. Tansu, “Optical gain and laser characteristics of InGaN quantum wells on ternary InGaN substrates,” *IEEE Photon. J.*, vol. 5, no. 2, Apr. 2013, Art. no. 2600111

*Example when using et al.:*

1. S. Azodolmolky *et al.*, Experimental demonstration of an impairment aware network planning and operation tool for transparent/translucent optical networks,” *J. Lightw. Technol.*, vol. 29, no. 4, pp. 439–448, Sep. 2011.

**First A. Author** (M’76–SM’81–F’87) and all authors may include biographies. Biographies are often not included in conference-related papers. This author became a Member (M) of IEEE in 1976, a Senior Member (SM) in 1981, and a Fellow (F) in 1987. The first paragraph may contain a place and/or date of birth (list place, then date). Next, the author’s educational background is listed. The degrees should be listed with type of degree in what field, which institution, city, state, and country, and year the degree was earned. The author’s major field of study should be lower-cased.

The second paragraph uses the pronoun of the person (he or she) and not the author’s last name. It lists military and work experience, including summer and fellowship jobs. Job titles are capitalized. The current job must have a location; previous positions may be listed without one. Information concerning previous publications may be included. Try not to list more than three books or published articles. The format for listing publishers of a book within the biography is: title of book (publisher name, year) similar to a reference. Current and previous research interests end the paragraph.

The third paragraph begins with the author’s title and last name (e.g., Dr. Smith, Prof. Jones, Mr. Kajor, Ms. Hunter). List any memberships in professional societies other than the IEEE. Finally, list any awards and work for IEEE committees and publications. If a photograph is provided, it should be of good quality, and professional-looking. Following are two examples of an author’s biography.

**Second B. Author** was born in Greenwich Village, New York, NY, USA in 1977. He received the B.S. and M.S. degrees in aerospace engineering from the University of Virginia, Charlottesville, in 2001 and the Ph.D. degree in mechanical engineering from Drexel University, Philadelphia, PA, in 2008.

From 2001 to 2004, he was a Research Assistant with the Princeton Plasma Physics Laboratory. Since 2009, he has been an Assistant Professor with the Mechanical Engineering Department, Texas A&M University, College Station. He is the author of three books, more than 150 articles, and more than 70 inventions. His research interests include high-pressure and high-density nonthermal plasma discharge processes and applications, microscale plasma discharges, discharges in liquids, spectroscopic diagnostics, plasma propulsion, and innovation plasma applications. He is an Associate Editor of the journal *Earth*, *Moon*, *Planets*, and holds two patents.

Dr. Author was a recipient of the International Association of Geomagnetism and Aeronomy Young Scientist Award for Excellence in 2008, and the IEEE Electromagnetic Compatibility Society Best Symposium Paper Award in 2011.

**Third C. Author, Jr.** (M’87) received the B.S. degree in mechanical engineering from National Chung Cheng University, Chiayi, Taiwan, in 2004 and the M.S. degree in mechanical engineering from National Tsing Hua University, Hsinchu, Taiwan, in 2006. He is currently pursuing the Ph.D. degree in mechanical engineering at Texas A&M University, College Station, TX, USA.

From 2008 to 2009, he was a Research Assistant with the Institute of Physics, Academia Sinica, Tapei, Taiwan. His research interest includes the development of surface processing and biological/medical treatment techniques using nonthermal atmospheric pressure plasmas, fundamental study of plasma sources, and fabrication of micro- or nanostructured surfaces.

Mr. Author’s awards and honors include the Frew Fellowship (Australian Academy of Science), the I. I. Rabi Prize (APS), the European Frequency and Time Forum Award, the Carl Zeiss Research Award, the William F. Meggers Award and the Adolph Lomb Medal (OSA).

1. This paragraph of the first footnote will contain the date on which you submitted your paper for review. It will also contain support information, including sponsor and financial support acknowledgment. For example, “This work was supported in part by the U.S. Depart­ment of Com­merce under Grant BS123456.”

   The next few paragraphs should contain the authors’ current affiliations, including current address and e-mail. For example, F. A. Author is with the National Institute of Standards and Technology, Boulder, CO 80305 USA (e-mail: author@ boulder.nist.gov).

   S. B. Author, Jr., was with Rice University, Houston, TX 77005 USA. He is now with the Department of Physics, Colorado State University, Fort Collins, CO 80523 USA (e-mail: author@lamar.colostate.edu).

   T. C. Author is with the Electrical Engineering Department, University of Colorado, Boulder, CO 80309 USA, on leave from the National Research Institute for Metals, Tsukuba, Japan (e-mail: author@nrim.go.jp). [↑](#footnote-ref-0)