UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

MATEUS FÉLIX JACOB

CONTROLLER AREA NETWORK APLICADA A UM BRAÇO ROBÓTICO

FORTALEZA

2019

MATEUS FÉLIX JACOB

CONTROLLER AREA NETWORK APLICADA A UM BRAÇO ROBÓTICO

Monografia apresentada ao curso de Engenharia Elétrica do Departamento de Engenharia Elétrica da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Dalton de Araújo Honório.
Co orientador: Prof. Dr. Antônio Barbosa de Souza Júnior.

FORTALEZA
2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

J16c Jacob, Mateus Félix.
Controller Area Network aplicada a um braço robótico / Mateus Félix Jacob. – 2019.
59 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia Elétrica, Fortaleza, 2019.
Orientação: Prof. Dr. Dalton de Araújo Honório.
Coorientação: Prof. Dr. Antônio Barbosa de Souza Júnior.

1. Protocolo de comunicação. 2. Protocolo CAN. 3. Sistemas Embarcados. 4. Braço robótico. 5. Microcontroladores. I. Título.

CDD 621.3

MATEUS FÉLIX JACOB

CONTROLLER AREA NETWORK APLICADA A UM BRAÇO ROBÓTICO

Monografia apresentada ao curso de Engenharia Elétrica do Departamento de Engenharia Elétrica da Universidade Federal do Ceará como requisito parcial à obtenção do título de Bacharel em Engenharia Elétrica.

Aprovada em: ___/___/_____.

BANCA EXAMINADORA

_____
Prof. Dr. Dalton de Araújo Honório (Orientador)
Universidade Federal do Ceará (UFC)

_____
Prof. Dr. Antônio Barbosa de Souza Júnior
Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE-Campus Maracanaú)

_____
Prof. Me. Josias Guimarães Batista
Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE-Campus Fortaleza)

_____
Me. Darielson Araújo de Souza
Universidade Federal do Ceará (UFC)

A Deus.

Aos meus pais, Rubens e Joselice.

À minha esposa, Nara.

AGRADECIMENTOS

À Deus, por ter me abençoado e ter me dado forças para superar todos os obstáculos.

À minha esposa, Nara, que sempre me acompanhou nessa trajetória de desafios e abdicações.

Aos meus pais, por terem me apoiado e incentivado a sempre estudar e terem me educado para a vida.

Às minhas irmãs que sempre tornavam, com suas personalidades, os momentos juntos mais agradáveis.

Aos Prof. Dr. Dalton Honório e Prof. Dr. Antonio Barbosa, pela excelente orientação, parceria e suporte. Sempre descomplicando os problemas e animando os dias.

Aos professores participantes da banca examinadora Prof. Me. Josias Guimarães Batista e Eng. Me. Darielson Araújo pelo tempo, pelas valiosas colaborações e sugestões.

Ao programa Ciências Sem Fronteiras que proporcionou muitos aprendizados acadêmicos e pessoais.

Aos colegas de graduação, Tiago Mota, Mateus Vieira, Emmanuel Abdala, Filipe Saraiva, Rômulo, Nestor, Matheus Kleming, José Oliveira, Túlio Naamã, pelas reflexões e bons momentos de descontração vividos.

"Não se deve ir atrás de objetivos fáceis, é preciso buscar o que só pode ser alcançado por meio dos maiores esforços."

Albert Einstein

RESUMO

Esse projeto tem como objetivo o desenvolvimento de um meio de comunicação entre DSPs responsáveis pelo controle de cada grau de liberdade de um braço robótico situado no laboratório GPAR-UFC. Essa comunicação é estabelecida através de uma rede CANbus interligando os DSPs os quais podem ser programados através dos softwares Code Composer Studio e Simulink. Fez-se uso da ferramenta Embedded Coder.

Palavras-chave: Protocolo de comunicação, Protocolo CAN, Sistemas Embarcados, Braço robótico, Microcontroladores.

ABSTRACT

This project aims to develop a means of communication between DSPs responsible for controlling each degree of freedom of a robotic arm located in the GPAR-UFC laboratory. This communication is established through a CANbus network interconnecting the DSPs which can be programmed through the Code Composer Studio and Simulink software. The Embedded Coder tool was used.

Keywords: Communication protocol, CAN protocol, Embedded Systems, Robotic Arm, Microcontrollers.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACK	*Acknowledge*
CAN	Controller Area Network
CCS	Code Composer Studio
CI	Circuito Integrado
CRC	*Cyclic Redundancy Check*
CSMA/CA	*Carrier Sense Multiple Access with Collision Avoidance*
DEL	*Delimiter*
DSP	Digital Signal Processor
GPAR	Grupo de Pesquisa em Automação e Robótica
GND	Ground (0V)
HIL	*Hardware in the Loop*
IoT	Internet of Things
OSI	Open System Interconnection
TI	Texas Instruments

SUMÁRIO

1 INTRODUÇÃO

Com o avanço da tecnologia, sistemas eletrônicos e eletromecânicos estão cada vez mais presentes em objetos e veículos motorizados e a necessidade da comutação de dados entre esses é cada vez maior. Tendo como objetivo melhorar a troca de informações entre vários equipamentos de controle, diminuindo a quantidade de cabos necessários para estabelecer tal troca nos sistemas eletrônicos em veículos, a fabricante Bosch™ desenvolveu, no final da década de 1980, o protocolo de comunicação serial *Controller Area Network* (CAN), no qual poderia utilizar de um a quatro fios como meio físico de comunicação. (1)

No princípio, foi muito aplicado em veículos automotores, assim mostrou sua eficiência e rapidez quanto a transmissão de dados e sua robustez na identificação e resolução de falhas. Dessa forma, vários outros protocolos baseados no CAN foram criados para aplicações específicas, por exemplo, *CANopen*® em sistemas embarcados, *DeviceNet*™ em processos industriais e *SAE J1939* em veículos e máquinas agrícolas entre outros exemplos.(2)

Nesse trabalho, será abordada a aplicação do protocolo de comunicação CAN para estabelecer um meio físico de comunicação bidirecional entre Processadores de Sinais Digitais (*Digital Signal Processors* DSP), utilizando um dos protocolos disponíveis pelo DSP. Além disso, a abordagem da comunicação entre os DSPs será refeita, mas, dessa vez, sendo programados a partir de um modelo no *software* Simulink®. Além do mais, guiado pelo desejo de estabelecer uma comunicação em tempo real entre um modelo do Simulink® e o barramento CAN, a utilização para tal fim da ferramenta *Desktop Real Time*® está retratada nesse trabalho.

A implementação de um meio físico de comunicação entre os DSPs e a programação desses DSPs através do *software* Simulink® foi alcançada com êxito. Entretanto, a comunicação em tempo real entre um modelo no Simulink® e o barramento CAN não foi alcançado devido a limitações dos computadores usados nos testes.

1.1 Origem do protocolo CAN

Na década de 1970, com a evolução da eletrônica, percebeu-se a tendência de substituição gradual de sistemas puramente mecânicos ou hidráulicos por sistemas eletrônicos. A evolução do hardware em conjunto com o software, possibilitava desenvolver subsistemas mais complexos que melhorariam a dirigibilidade do motorista, por exemplo,

subsistemas de freio antitravamento (*Anti-block Bracking System* ABS), de controle de estabilidade, de controle de torque, de direção elétrica assistida, de suspensões ativas e de controle do motor. Além disso, melhorias em relação ao controle de dispositivos como os subsistemas de faróis, de climatização, de áudio entre outros, seriam possíveis de ocorrerem.(3)

Dessa forma, com o desenvolvimento da tecnologia, a troca de informação entre as unidades de controle (*Electronic Control Units* - ECUs), responsáveis pelo controle de cada subsistema, tronava-se cada vez maior. Entretanto, nessa época, a comunicação entre sistemas eletrônicos embarcados era do tipo ponto-a-ponto, tornando-se inviável no caso de estabelecer comunicação entre um número elevado de ECUs, visto que se cada ECU estabelecer uma conexão com as demais, a quantidade de conexões resultante cresce em um fator quadrático.(3)

Tendo em vista os problemas de acréscimo de peso, de custo, de complexidade e de confiabilidade em uma comunicação ponto a ponto entre inúmeras unidades eletrônicas, empresas passaram a desenvolver protocolos multiplexados que simplificariam o meio físico de comunicação. (3)

Com esse propósito, a Bosch™ desenvolveu o protocolo *Controller Area Network* (CAN) na década de 1980, sendo padronizado pelas normas *ISO 11898* e *CAN 2.0 Specification* no início da década de 1990.

O primeiro veículo a ser produzido com um sistema baseado no protocolo CAN, foi o Mercedes-Benz W140, e com o passar dos anos foi intensamente utilizado em diversos modelos de vários fabricantes. (3)

Desde então, esse protocolo foi cada vez mais usado nas mais diversas áreas de aplicação, resultando no surgimento de novos protocolos mais completos baseados no CAN para aplicações mais específicas.

1.2 Áreas e aplicações do CAN

À medida que a tecnologia avança, sistemas embarcados de controle digital, oriundos de sistemas de controle analógicos, transformam-se gradativamente em sistemas de controle distribuído os quais fazem utilização de redes *field bus*, isto é, redes industriais para controle distribuído em tempo real.

Sendo um dos primeiros protocolos criados conforme o conceito de controle distribuído aplicado em barramentos de campo (*field bus)*, o CAN apresenta alta performance e alta confiabilidade, o que o torna bem cogitado para as mais diversas áreas de aplicação.

O novo padrão *CANbus extension protocol*, criado em 2015 pela *European Cooperation for Space Standadization* (ECSS), preenche as lacunas existentes na primeira padronização, *CAN Specification 2.0*, relacionadas a aplicações em espaçonaves. Este novo padrão, entre outras novidades, regulamenta o uso do protocolo genérico, CANopen®, da camada de aplicação (*application layer*) em sistemas de comunicação em espaçonaves.

Objetivamente, como a documentação *CAN Specification 2.0* estabelece regras somente para as camadas físicas (*physical layers*) e as camadas de ligação (*link layers*), o protocolo CAN torna-se cada vez mais presente em aplicações aeroespaciais, mais especificamente, em sistemas de comunicação de manipuladores espaciais. Explicações a respeito das camadas (*layers*) de implementação de comunicação baseada em CAN serão abordadas mais adiante. (4)

CANopen® também pode ser encontrado em aplicações de comunicação em braços robóticos, em veículos militares, em robôs industriais, em equipamentos médicos, em elevadores, em maquinário de produção e de empacotamento, em escavadeiras, em tratores entre outros. (8)

Outro protocolo baseado em CAN atuante em nível da camada de aplicação (*application layer*) é o SAE J1939, normalizado pela Sociedade de Engenheiro Automotivos (*Society of Automotive Engineers* SAE). Estão presentes em aplicações como veículos militares, máquinas e veículos agrícolas, veículos pesados (caminhões e ônibus), máquinas de construção com sistemas hidráulicos, navios e até mesmo na geração de energia. As perspectivas para esse protocolo são de crescimento no número de aplicações envolvendo o conceito de Internet das Coisas (*Internet of Things* IoT) e mobilidade conectada, em que os veículos se comunicarão entre si, usando registradores de dados via internet sem fio (*Wifi data loggers*). (6)

Originalmente desenvolvido no início da década de 1990 pela então Allen Bradley®, adquirida posteriormente pela Rockwell Automation®, o protocolo DeviceNet é amplamente empregado em processos industriais automatizados. Baseado também em CAN, atua à nível da cama de aplicação (*application layer*). Sua aplicação pode ser percebida nos mais diversos ramos de processos industriais, por exemplo nas indústrias alimentícias, indústrias químicas e petroquímicas, indústrias têxteis, entre outras. (7)

Como pode-se perceber, é um protocolo de frequênte uso nas mais diversas áreas de atuação, fazendo uso dos protocolos de uso específico e mostrando sua confiabilidade em diversos ambiente de aplicação.

1.3 CAN e Internet das Coisas (IoT)

A expansão do acesso à internet no cotidiano das pessoas, por meio de *tablets*, de *smartphones*, de computadores portáteis, de redes sociais, transformou o perfil do consumidor. Concomitante a isso, a necessidade de melhoria de produtos e serviços ofertados no mercado, impulsionou a tendência de digitalização de processos de fabricação em empresas e em indústrias. Em meio a esse cenário, surge um conceito Internet das Coisas (*Internet of Things* IoT), que tem como base a ideia de todos os dispositivos estarem conectados, enviando dados a um banco de dados, onde poderão ser analisados para diversos fins, podendo melhorar o embasamento para a tomada de decisão e aumentando a eficiência e a qualidade do serviço ou do produto. (9)

Atualmente, com o surgimento do conceito Internet das coisas (*Internet of Things* IoT), algumas empresas desenvolvedoras de tecnologia já estão a produzir equipamentos de instrumentação que se comunicam via CAN e armazenam seus dados em um *Big Data*, aplicando conceitos de IoT. Entretanto, normalmente, aplicações mais complexas de IoT requerem uma transferência de grandes dados, aumentando a gama de possíveis informações que possam ser transmitidas. Assim, como o CAN é um protocolo que envia dados com tamanho máximo de 8 *bytes*, são restritas as aplicações do conceito IoT com protocolo CAN, devido a essa limitação do tamanho do dado.

Em contra partida, devido a intensa presença do protocolo *Ethernet* em áreas profissionais, áreas públicas e áreas residenciais, a fabricação de componentes eletrônicos referentes a utilização do *Ethernet* sofreu um severo crescimento, resultando em uma diminuição do custo dessa tecnologia, em vista do aumento da produção em escala desses componentes. Além disso, o protocolo *Ethernet* é capaz de transferir grandes pacotes de dados, sendo de 42 bytes o menor pacote, a uma taxa de 100Mbps. Inclusive o número de dispositivos conectados a uma rede *Ethernet*, geralmente, é ilimitado. (10)

Desse modo, em ambientes fabris de automação, as redes de campo (*field bus*) estão perdendo espaço para as redes *Ethernet* em aplicações. Habitualmente, é requisitado dessas aplicações, a necessidade de garantir um comportamento em tempo real nas redes

compostas por inúmeras máquinas, dentre estas, máquinas com alto nível de transferência de dados. Isso ocorre pelo fato de o protocolo CAN transmitir pacotes de, no máximo, 8 *bytes* a uma taxa de 1Mbps. (10)

Em aplicações IoT, normalmente, a quantidade de equipamentos conectados que requerem uma elevada troca de dados é um fator crucial na escolha do uso do protocolo *Ethernet*, mesmo tendo um custo de implementação mais elevado que se fosse com protocolo CAN. (9)

Entretanto, em 2012, foi apresentado o protocolo CAN FD, com capacidade para transferir dados de até 64 bytes a uma taxa máxima de 15Mbps para distâncias de até 250m. Esse novo protocolo, continua reunindo todas as qualidades do tradicional CAN versão 2.0, mas agora podendo ser usado em aplicações IoT em tempo real com transferência de maiores dados. (10)

Atualmente, como não existem muitos CIs fabricados integrando o protocolo CAN FD, esse ainda não está tão disseminado nas mais diversas áreas como o protocolo tradicional CAN 2.0. Em alguns anos, a perspectiva de crescimento de aplicações desse protocolo é elevada, pois é perfeito em sub redes para aplicações IoT, visto que, dentro de máquinas, será possível o manejo de dados de automação e de controle, paralelamente, aos dados referentes a IoT, resultando em economia de custo. (10)

Mesmo havendo a restrição da do tamanho dos dados no protocolo CAN 2.0, há a possibilidade de realizar aplicações restritas de IoT, por exemplo, em máquinas ou em braços robóticos, utilizando o protocolo tradicional nos seus formatos padrão e estendido. Isso é possível por meio do uso de placas de desenvolvimento, por exemplo, CAN32, ESP8266, ESP32 entre outras.

Em aplicações com a placa de desenvolvimento CAN32, esta pode ser conectada diretamente ao barramento CAN, coletando os dados da rede e enviando-os diretamente, via internet sem fio (*Wi-fi*), para a nuvem (*Cloud*) no intuito de analisar os dados com inteligência artificial. Já em aplicações com as placas de desenvolvimento ESP8266 e ESP32, estas podem ser conectadas a um barramento CAN por meio de um transceptor CAN (CAN *Transceiver*). As informações adquiridas via esse barramento, podem ser enviadas, via sinal de Internet sem fio (*Wifi*), para uma Núvem (*Cloud*) e analisadas com inteligência artificial. (11)

Como auxílio às placas de desenvolvimento citadas, é aconselhável o uso de analisador lógico, possibilitando visualizar os sinais no barramento CAN, facilitando a detecção de falhas. Além disso, a comunicação via Internet sem fio (*Wifi*) pode ser auxiliada pelo *software* Firebase™ desenvolvido pela Google™.

1.4 Escolha do protocolo CAN

A placa de desenvolvimento eZdsp F28335 disponibiliza os protocolos *Serial Communication Interface* (SCI ou UART), *Serial Peripheral Interface* (SPI), *Inter-Integrated Circuit* (I2C), *Enhanced Controller Area Network* (eCAN). Abaixo, algumas características desses protocolos são mostradas, ajudando na escolha do protocolo.

O protocolo SPI, é normalmente usado em aplicações on-board para conectar dois circuitos integrados (CIs) ou periféricos (ADC, EEPROM, Displays etc), estabelece uma comunicação serial *full-duplex* baseada no conceito mestre/escravo cuja utilização de quatro fios é necessária para criar o barramento SPI (SPI-Bus) de comutação de dados. Esse protocolo é indicado para envio de grande quantidade de dados entre pequeno número de periféricos. Esse protocolo pode atingir uma taxa máxima de transferência de 10Mbps. Entretanto, para distâncias maiores que um metro, não conseguem comutar dados entre o mestre e o escravo com qualidade. Além disso, a necessidade de fios adicionais para se comunicar com múltiplos escravos através do mesmo meio físico SPI-Bus é uma desvantagem para redes contendo vários dispositivos. (14)

O protocolo I2C, habitualmente usado em aplicações embarcadas para estabelecer comunicação entre CIs ou periféricos (ADC, EEPROM, Displays etc), baseia-se em uma comunicação serial *half-duplex*, em que o conceito multi-mestre está presente, e necessita somente de dois fios para estabelecer o seu barramento I2C (I2C-Bus) de transferência. Com isso, é indicado para envio de pequeno número de dados para vários periféricos. Todavia, não são recomendados para transferência de dados a grandes distâncias. Esse protocolo pode comutar a uma taxa máxima de 400Kbps, apresentando facilidade para adicionar mais periféricos ao I2C-Bus. (15)

O protocolo SCI (UART), destinado a função de troca de dados entre dois dispositivos, não possibilita a criação de uma rede de comunicação entre os eZdsps como desejado para esse trabalho. (18)

O protocolo CAN, destinado a aplicações veiculares, consegue transferir dados entre periféricos em um meio físico composto de um a quatro fios (*Barramento CAN*). Baseado em comunicação serial multi-mestre *half-duplex*, permite a fácil adição de novos periféricos à rede CAN-Bus. Diferente dos protocolos seriais acima citados, o CAN faz uso do princípio produtor-consumidor, em que uma mensagem não possui um endereço de destino, mas sim um endereço do parâmetro, por exemplo tensão e corrente do motor. Assim, uma mensagem transmitida por um dispositivo é recebida por todos os demais dispositivos,

cabendo a estes verificarem se o endereço da mensagem indica que é importante para o dispositivo. Além disso, até uma distância de 40m, esse protocolo pode comutar dados a uma taxa máxima de 1Mbps. Entretanto, 1200m é a máxima distância para viabilizar uma comunicação a uma taxa de transmissão aceitável, sendo que quanto maior for essa distância, menor será a máxima taxa de transferência possível de ser obtida. (19)

Assim, visando o objetivo de utilizar o que a placa eZdsp F28335 já disponibiliza, o protocolo CAN (chamado de eCAN no eZdsp F28335) foi escolhido, pois necessita de poucos fios para estabelecer um meio físico de comunicação entre os DSPs, podendo apresentar um barramento CAN que alcance 40m de comprimento, mantendo sua taxa de transmissão em seu valor máximo de 1Mbps. Além disso, como é um sinal diferencial entre dois outros sinais, mesmo que sofra uma interferência magnética e suas tensões variem, a diferença entre as tensões permanece a mesma.

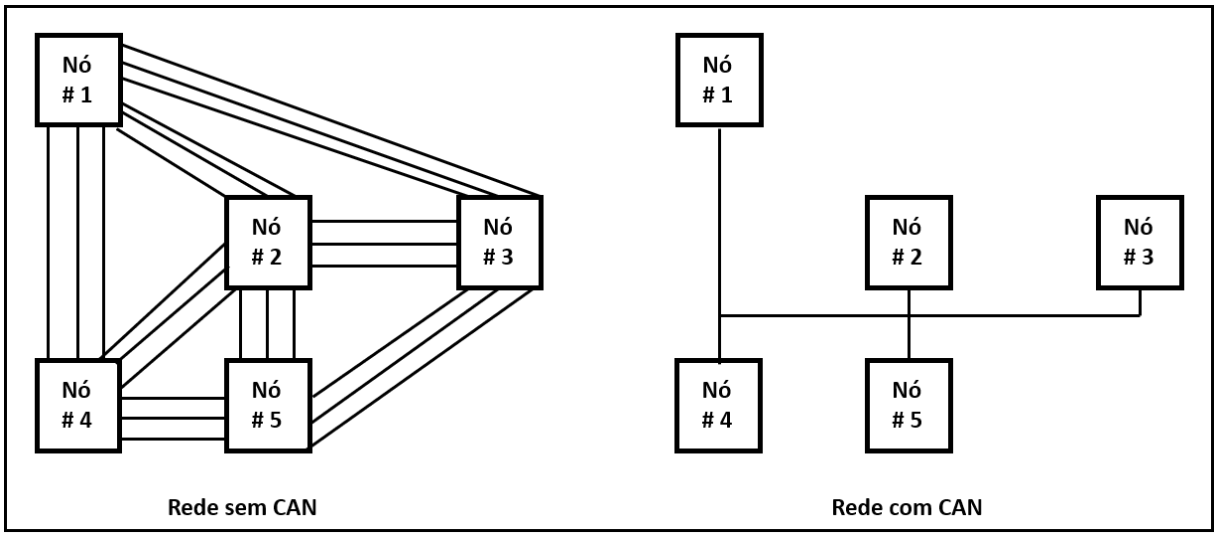
Uma outra vantagem desse protocolo é a presença de uma ferramenta de verificação de redundância cíclica (*Cyclic Redundancy Check* CRC) além dos registradores convencionais para identificação de erros. Através dessa ferramenta, um valor é calculado a partir dos dados da mensagem armazenada no controlador CAN e da mensagem enviada no barramento CAN. Se esses dois valores calculados forem iguais, significa que a mensagem no barramento é igual a mensagem armazenada no controlador.

Uma outra característica que influenciou a escolha foi o fato de ser um protocolo serial multi mestre que, ao contrário dos demais protocolos, o endereçamento é feito por mensagem e não por dispositivo de destino. Isso resulta na possibilidade de um dispositivo enviar a mesma mensagem a um único dispositivo ou a todos os dispositivos de uma só vez.

Além disso, como é uma tecnologia do protocolo CAN bem difundida no mercado, circuitos integrados e placas de desenvolvimento que utilizam esse protocolo para auxiliar a conexão de outros dispositivos a um barramento CAN estão cada vez mais baratos. Assim, o custo para expansão da rede CAN pode ser de baixo custo.

Na Figura 1, pode-se verificar a simplificação das conexões entre os dispositivos ao utilizar o protocolo CAN, devido a diminuição da quantidade de cabos utilizados para estabelece o meio físico de comunicação.

Figura 1 – Complexidade de conexões em rede de comunicação com e sem CANbus.



Fonte: COPPERHILL TECHNOLOGIES® (2019).

Na Tabela 1, pode-se verificar como a taxa transmissão varia com o aumento do comprimento do barramento. Assim, o comprimento do barramento CAN no braço robótico, pode chegar a um comprimento de 40 metros mantendo sua máxima taxa de transferência de dados.

Tabela 1 – Taxas de transmissão e suas distâncias máximas.

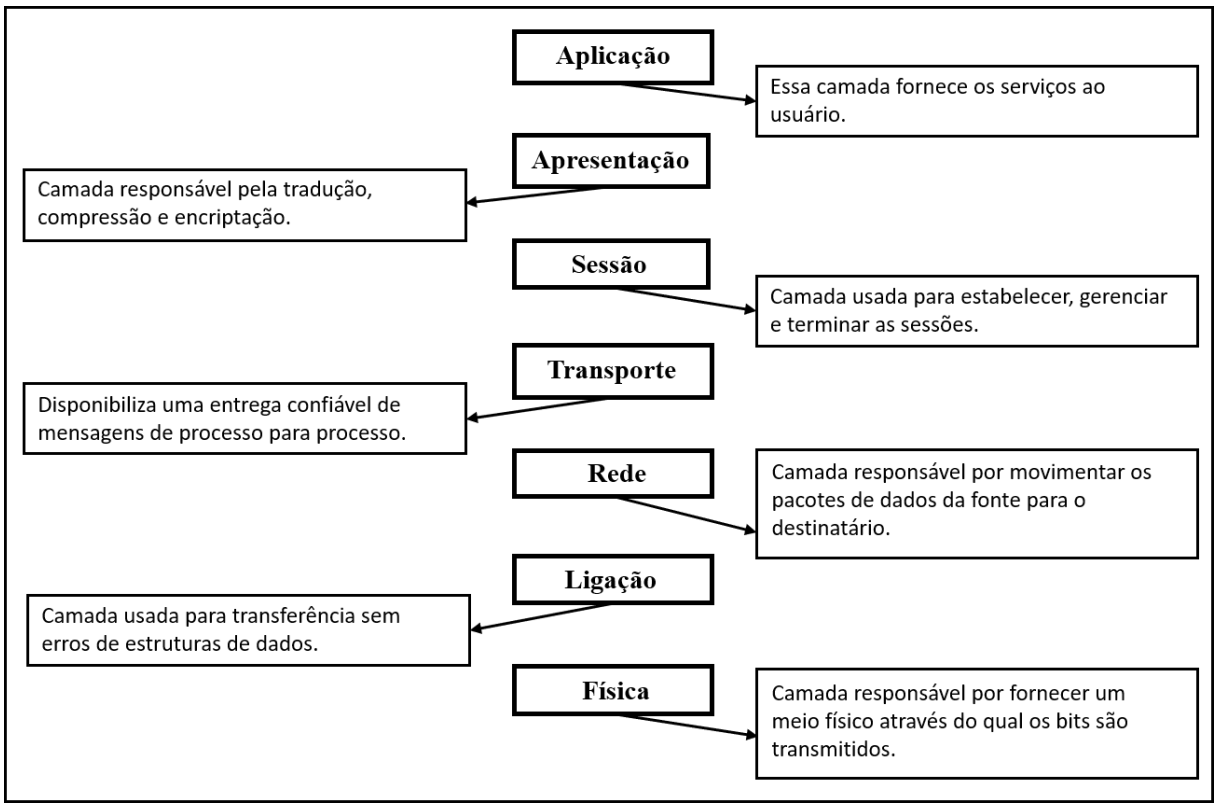
Taxa de transmissão (Kbps)	Tempo de transmissão por bit (µs)	Comprimento do barramento (m)
1000	1	30
800	1,25	50
500	2	100
250	4	250
125	8	500
62,5	16	1000
20	50	2500

Fonte: JAVA T POINT (2019).

1.5 Principais camadas do modelo OSI para protocolo CAN e seus derivados

O modelo *Open System Interconnection* (OSI) é uma estrutura de sete camadas em que se baseiam vários padrões de rede de comunicação. As três primeiras camadas são chamadas de camadas inferiores e as quatro últimas, de camadas superiores. (19) Na figura 2, todas as camadas pertencentes ao modelo OSI estão ilustradas. Esse trabalho apresenta um foco maior nas duas primeiras e na última camada.

Figura 2 – Camadas (layers) do modelo OSI.



Fonte: JAVA T POINT (2019).

A primeira, camada física, engloba o meio físico e suas características, por exemplo, os cabos conexão, os conectores usados, a distância máxima permitida para transmissão, o modelo de transmissão *half-duplex*, a topologia de conexão da rede, o tipo de sinal utilizado entre outros.

A segunda, camada de ligação, é a camada de mais baixo nível em que são atribuídos significados aos bits transmitidos, assim, é encarregada pela transferência de pacotes de dados na rede de comunicação. É responsável, também, pela interpretação dos sinais analógicos transferidos através da camada física, pela formação do pacote de dados

(*frame*), pelo controle do fluxo dados, pelo controle de erro através do cálculo do valor de verificação de redundância cíclica (CRC).

A última, camada de aplicação, utiliza interfaces de programação para solicitar serviços de rede. Nesta camada, o usuário final tem acesso a serviços de rede.

2 PROTOCOLO CAN E SUAS CARACTERÍSTICAS

O protocolo CAN está muito presente no mercado devido suas características de conexão entre os dispositivos, além dos vários mecanismos para detecção de erro na transmissão da mensagem.

O conhecimento do modo de operação desse protocolo, das ferramentas de detecção de erros que utiliza, dos possíveis tipos de mensagens que podem ser encontrados no barramento é de suma importância para compreender como o protocolo opera para transmitir e receber mensagens, para saber qual o nível de controle dos parâmetros de cada tipo de mensagem o programador possui e para identificar como interpretar as mudanças ocorridas nos registradores de estado e de erro para identificar e prevenir que erros ocorram novamente.

Nesse capítulo, serão abordadas a arquitetura do protocolo CAN, o formato do sinal no barramento, seus principais registradores e suas funções, os possíveis tipos de mensagem que podem ser enviadas ao barramento, além de uma explicação dos principais erros que podem ocorrer.

2.1 Arquitetura do módulo eCAN

Uma rede CAN é formada por vários dispositivos conectados ao mesmo barramento de comunicação. A troca de informação ocorre devido a utilização, por dispositivo, de uma unidade controladora CAN (*CAN controller*) e de um transceptor CAN (*CAN transceiver*) que adequam o formato dos dados presentes na Unidade Central de Processamento (CPU) ao formato que possibilita a leitura dos dados por outros dispositivos conectados à rede. A arquitetura do CAN está mostrada na figura 3.

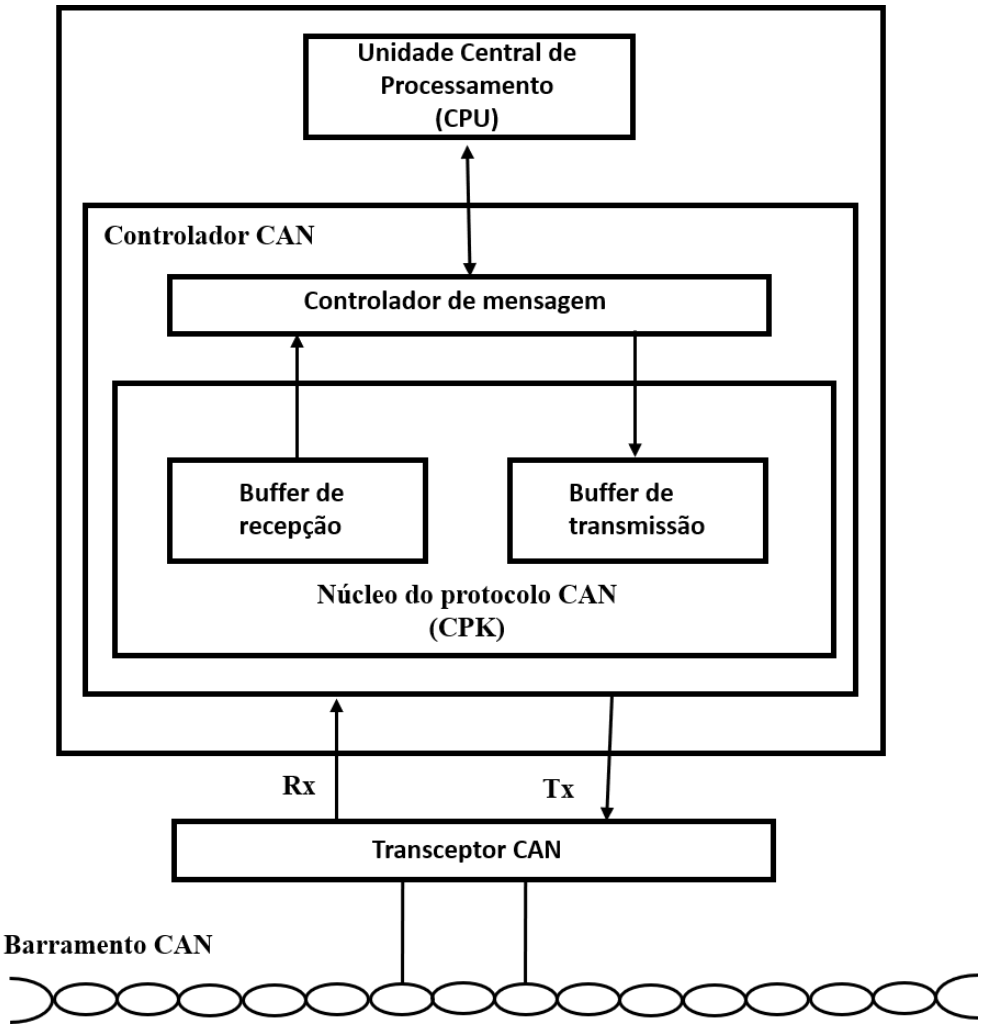
Cada controlador CAN possui 32 caixas de correio (*mailboxes*), registradores de configuração e de controle, registradores de status, unidade temporizadora, unidade de controle de recepção e uma interface com CPU. Mais especificamente, a função do controlador CAN é realizar a comunicação entre o módulo CAN e a CPU do DSP, configurar os registradores de CPU para o funcionamento desejado, gerenciar o temporizador para ser possível obter o tempo do envio da mensagem, estruturar a mensagem nos formatos padrão e estendido, entre outras.

Assim, no caso de uma recepção de mensagem, cabe ao Núcleo do Protocolo CAN (CPK) determinar se o dispositivo vai capturar a mensagem decodificada e armazenada

no *buffer* de recepção, oriunda da leitura do sinal no barramento, com o objetivo de utilizá-la na lógica presente na CPU. Já no caso de transmissão de uma mensagem, é dever do controlador CAN reunir o endereço da mensagem e o dado a ser transmitido e estruturá-los conforme os dois formatos permitidos, padrão e estendido.

Os sinais originados no controlador CAN não são adequados para a conexão direta ao barramento CAN. Para isso, faz-se uso de um circuito integrado transceptor CAN. No caso da transmissão, sua função é traduzir o formato da mensagem presente no controlador CAN para um sinal diferencial elétrico a ser transmitido no barramento. O mesmo processo, no sentido inverso, ocorre em uma recepção de mensagem, onde o sinal diferencial lido no barramento é traduzido para o formato da mensagem do controlador CAN.

Figura 3 – Arquitetura do módulo eCAN.



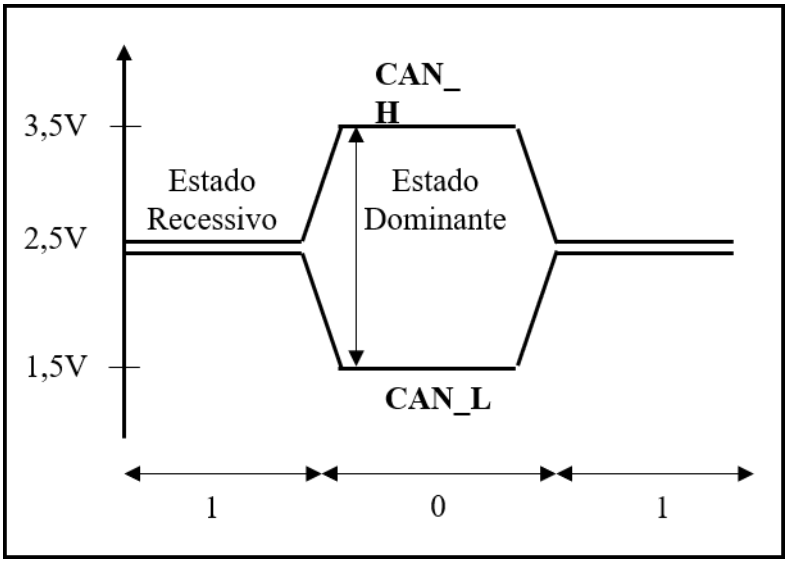
Fonte: TEXAS INSTRUMENTS™ (2019).

2.2 Formato dos sinais e conexões no barramento CAN

O dispositivo transceptor CAN gera dois sinais semelhantes e invertidos, CANH e CANL, que se baseiam em lógica dominante e recessiva. Como mostrado na figura 4, os sinais CANL e CANH são ondas quadradas complementares que variam de 1,5V a 2,3V e de 2,5V a 3,5V respectivamente. (21)

Segundo o padrão CAN, o estado lógico é formado pela diferença entre os sinais CANH e CANL. Dessa forma, o estado lógico é classificado como recessivo, simbolizado pelo número binário 1, quando a diferença entre os sinais CANH e CANL resulta em valores menores que 0,5V. Em contra partida, o estado lógico é classificado como dominante, simbolizado pelo número binário 0, quando a diferença entre os sinais atinge valores maiores que 0,5V.

Figura 4 – Estado lógico recessivo e dominante.



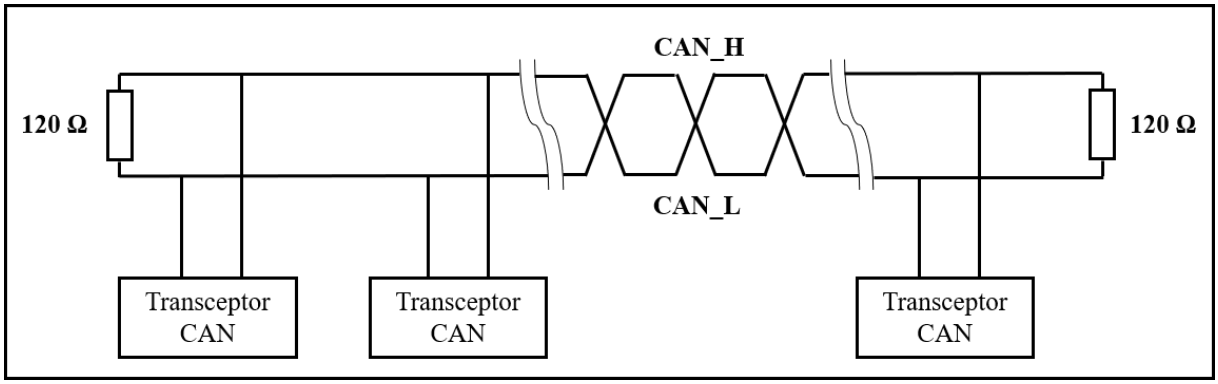
Fonte: DI NATALE (2012).

Para estabelecer a comunicação entre os dispositivos, faz-se necessário um barramento formado por um par de fios trançados, onde dois resistores terminais de 120 Ω são conectados em cada uma das duas extremidades do par trançado, como mostrado na figura 5.

A presença desses resistores terminais tem como função amenizar a ocorrência da reflexão do sinal, mantendo os valores de tensão no barramento dentro do intervalo permitido.

Quando não existem esses resistores terminais conectados ao barramento, os valores das tensões variam fora do intervalo permitido, mensagens de erro são enviadas ao barramento e os dispositivos são isolados do barramento defeituoso. (32)

Figura 5 – Conexão de dispositivos a rede CANbus.

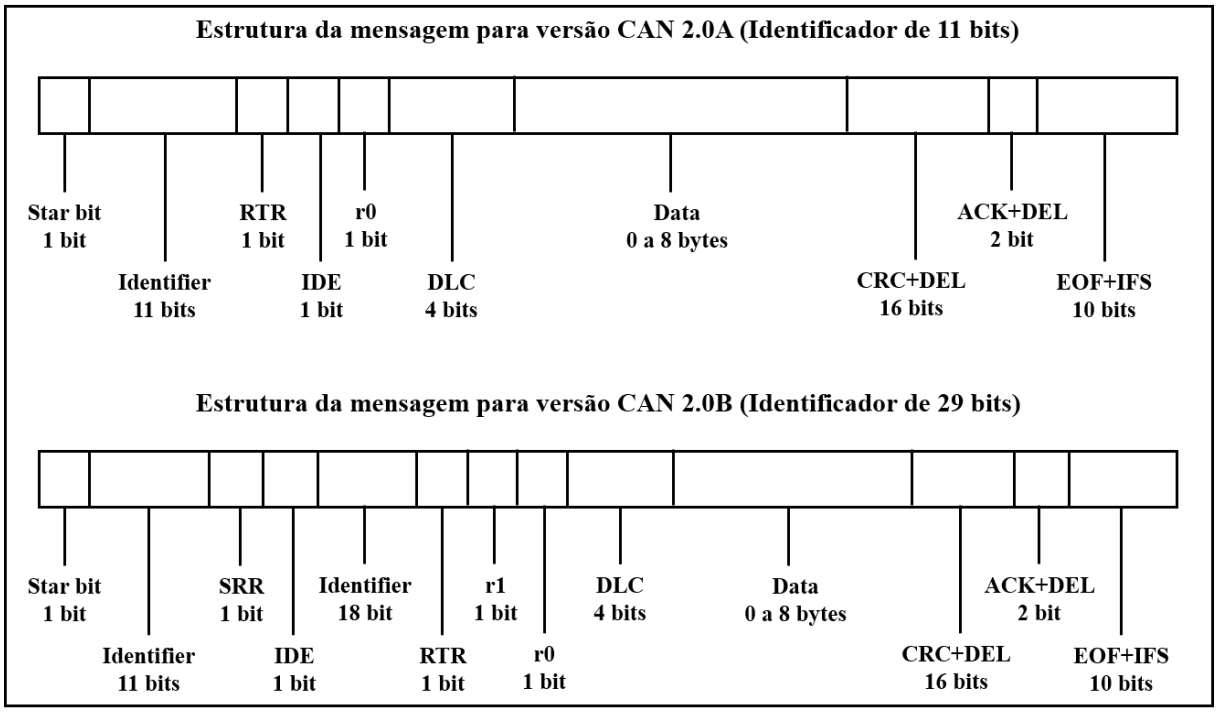


Fonte: TEXAS INSTRUMENTS™ (2008)

2.3 Formação das mensagens do Protocolo CAN

O dado a ser transmitido pode ser estruturado em dois formatos. Na figura 6, é possível perceber a semelhança entre eles de tal maneira que a versão CAN 2.0B (protocolo CAN estendido) possui os mesmos registradores e funções da versão CAN 2.0A (protocolo CAN padrão) e mais alguns outros registradores a possibilitar novas funcionalidades.

Figura 6 - Estrutura padrão e estendida da mensagem em protocolo CAN.



Fonte: FRATICELLI (2013).

2.3.1 Versão padrão da mensagem do protocolo CAN

A transferência de um dado, conforme protocolo CAN padrão (versão 2.0A), inicia-se após uma sequência mínima de 11 bits recessivos no barramento CAN (situação de barramento ocioso), indicando que o mesmo está apto para a transmissão de uma nova mensagem. Segundo o protocolo CAN padrão, a mensagem começa com um bit dominante, bit de início (*start bit*), para indicar o início da mensagem.

Em sequência, existe o campo de arbitragem (*arbitration field*), composto por 11 bits de identificação (*identifier bits*) e um bit de requisição de transmissão remota (*remote transmition request* RTR). Assim, os onze bits de identificação possibilitam a existência de 2048 identificadores de mensagens e informam qual parâmetro está a enviar, por exemplo, torque do motor, peso de uma peça, corrente elétrica do motor. Outra função dos bits de identificação é determinar a prioridade da mensagem, conforme a relação de quanto menor for o identificador, maior a prioridade da mensagem. Além disso, o bit RTR indica se essa mensagem será uma mensagem de dado (*data message*) ou uma mensagem de resposta (*reply message*). Esses tipos de mensagens serão explicados posteriormente.

Em seguida, existe o campo de controle (*control field*), formado pelo bit de extensão do identificador (*Identifier Extension* IDE), pelo bit reservado r0 e pelos quatro bits de código do tamanho do dado (*data length code* DLC). Dessa forma, o bit IDE determina qual versão do protocolo CAN, CAN2.0A ou CAN2.0B, a mensagem está moldada. Nesse caso da versão padrão, sempre apresenta valor dominante. Depois, o bit r0 é somente um bit reservado e não tem função. Além disso, os quatro bits DLC detêm a função de determinar quantos bytes possui o dado a ser transmitido, podendo assumir valores inteiros de zero a oito. Assim, o espaço é reservado para esse dado a ser transmitido conforme o valor indicado em DLC.

Logo após, encontra-se o campo de dado (*data field*), podendo armazenar um dado de zero a oito bytes. Normalmente, sua transmissão começa pelo *byte* mais significante.

Em sequência, há o campo de código de redundância cíclica (*CRC field*), integrando os quinze bits do CRC mais um bit delimitador do CRC (*delimiter* DEL). O CRC é usado para verificar se todos os bits dos campos de arbitragem, de controle e de dado presentes na mensagem transmitida são iguais aos bits recebidos referentes aos mesmos campos. Para o cálculo do valor a ser armazenado nos quinze bits CRC, faz-se uso da lógica

XOR, realizando uma divisão em que o resto dessa divisão será o número guardado nesses quinze bits CRC.

Em sequência, existe o campo de reconhecimento (*acknowledge field* ou *ACK field*), formado pelo bit de reconhecimento (*acknowledge* ACK) e pelo bit delimitador do ACK (*delimiter* DEL). O bit de reconhecimento pode assumir valor dominante, caso a mensagem esteja sendo transmitida por uma *mailbox* de transmissão, e pode assumir valor recessivo, caso a mensagem esteja sendo recebida por uma *mailbox* de recepção. O valor do bit delimitador sempre será recessivo para indicar o fim desse campo.

Em seguida, há o campo fim da mensagem (*end of frame field* EOF), formado por sete bits recessivos para indicar o fim da transmissão de uma mensagem. Somente nesse campo, a regra do *bit stuff* é intencionalmente violada, assim não gerando um erro por essa ação.

A regra do *bit stuff* obriga a impossibilidade de haver seis ou mais bits dominantes (valor 0) consecutivos durante a transmissão completa da mensagem. O controlador CAN é responsável por inserir e remover esses *bits stuff* e o programador não tem permissão para inserir ou retirar esses bits. Somente por meio do uso de osciloscópio e de analisadores lógicos, esses bits podem ser vistos.

Por fim, o campo espaço entre mensagem (*inter frame space* IFS), é composto por três bits com estados recessivos e tem o objetivo de garantir um tempo suficiente para finalizar o processamento interno do controlador CAN. Pode ser seguido da condição de barramento ocioso (*bus idle*) caso não haja nenhuma mensagem pendente a ser transmitida ou pode ser seguido de uma mensagem de dado ou de requisição caso haja alguma pendente.

2.3.2 Versão estendida da mensagem do protocolo CAN

A mensagem, conforme o protocolo CAN estendido (versão 2.0B), apresenta uma estrutura muito parecida com a versão padrão, diferenciando-se pela adição de alguns bits específicos.

Nessa versão, no campo de arbitragem (*arbitration field*), são acrescentados mais 18 bits no identificador, totalizando 29 bits para identificação de mensagens. Nesse caso, aproximadamente 536 milhões de identificadores de mensagens podem ser obtidos. Além disso, nesse mesmo campo, o bit solicitação remota substituta (*substitute remote request* SRR)

sempre apresenta um estado lógico recessivo, tendo função de manter a compatibilidade entre as versões CAN2.0A e CAN 2.0B.

O bit IDE, que na versão padrão fazia parte do campo de controle, na versão estendida, faz parte do campo de arbitragem. Para a versão estendida, esse bit sempre apresenta um valor recessivo. Além disso, existe um bit reservado r1 para uso futuro caso haja expansão do protocolo.

Os restantes dos campos são idênticos aos da versão padrão.

2.4 Arbitragem e prioridades de envio

O controlador CAN faz uso da ferramenta *Carrier Sense Multiple Access with Collision Avoidance* (CSMA/CA) para evitar colisão entre duas mensagens enviadas simultaneamente por dois ou mais dispositivos ao barramento CAN. O CSMA/CA atua somente no campo de arbitragem, analisando os identificadores das mensagens.

A arbitragem é uma das funções dessa ferramenta e tem o dever de comparar, bit a bit, o conjunto dos bits de identificação. Dessa forma, o bit dominante sempre prevalece na transmissão. Então mensagens com maior prioridade não têm suas transmissões atrasadas, obtendo a capacidade de execução em tempo real. Assim, conclui-se que quanto menor for o valor do identificador, maior será a prioridade.

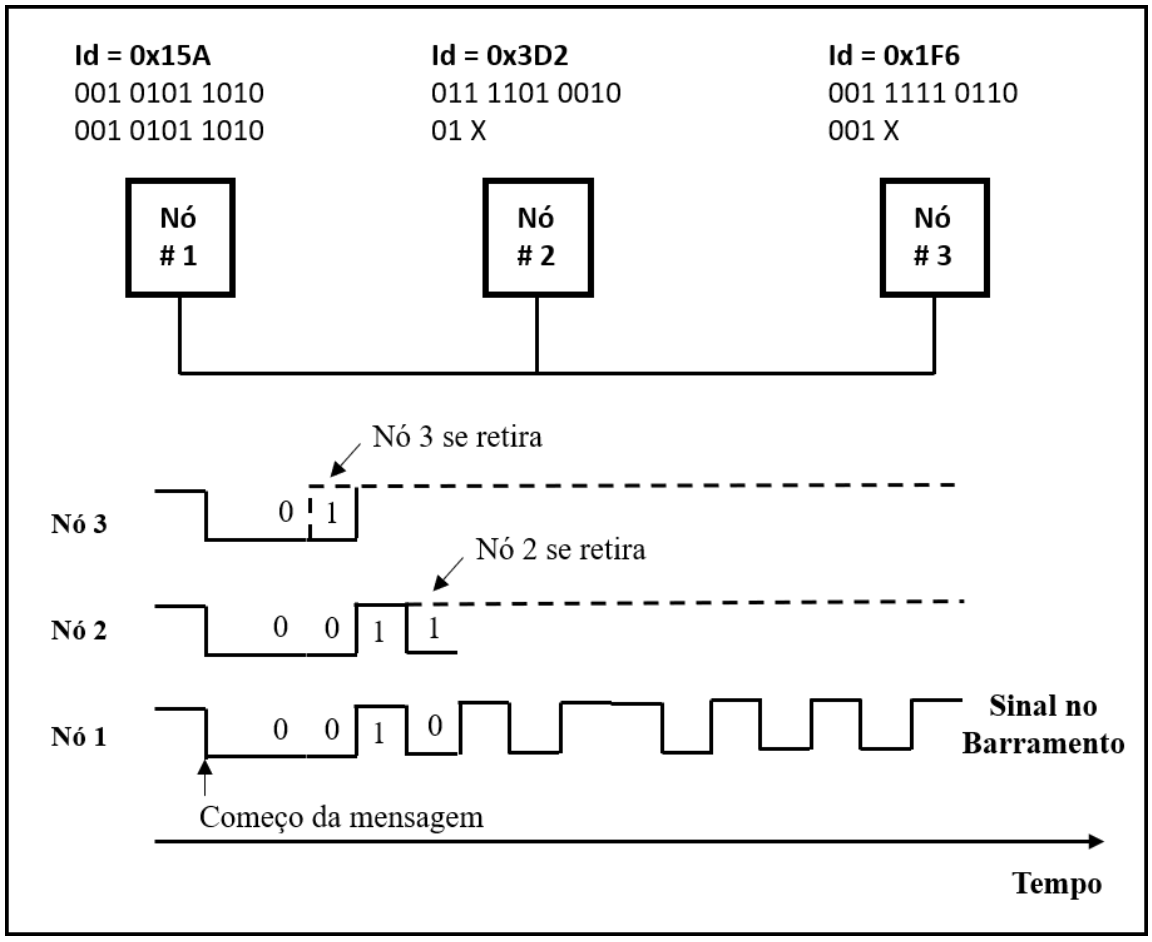
Devido as características dos transceptores CAN, todos os dispositivos conectados ao barramento CAN leem os dados da rede constantemente, incluindo os dados na fase de arbitragem.

Na figura 7 abaixo, uma exemplificação da atuação da ferramenta CSMA/CA onde três dispositivos iniciam, simultaneamente, uma transmissão de uma mensagem ao barramento CAN. Essa ferramenta compara, bit a bit, todos os bits do identificador da mensagem, fazendo prevalecer o estado dominante caso esteja presente. Desse modo, o resultado da arbitragem vai se formando com a mensagem do identificador de maior prioridade.

Dessa forma, verifica-se que cada dispositivo conectado ao barramento possui seu próprio endereço de mensagem e vai transmiti-la ao mesmo instante que os demais. Nesse caso, todos começam pelo bit de início no estado dominante. Em seguida, a ferramenta CSMA/CA começa a atuar na comparação de cada bit do campo de arbitragem enviado pelos dispositivos ao barramento. Assim, o primeiro bit enviado ao barramento pelos dispositivos é

de estado lógico dominante, logo o barramento CAN permanece em estado lógico dominante. Logo após, o segundo bit dos identificadores é enviado por cada um, mas nesse caso, o dispositivo 3 é excluído dessa comparação, pois seu bit é recessivo, prevalecendo o estado lógico dominante dos demais sinais no barramento CAN. Depois, a comparação continua entre os dispositivos 1 e 2, resultando em estado lógico recessivo no barramento, devido os dois apresentarei estado lógico recessivo. Entretanto, ao comparar o estado lógico do bit seguinte para os dois dispositivos, percebe-se que o estado lógico dominante prevalece, o dispositivo 2 é excluído da comparação e a mensagem do dispositivo 1 será transmitida no barramento CAN.

Figura 7 - Aplicação CSMA-CA.



Fonte: DI NATALE (2012).

Além disso, para o protocolo CAN padrão (versão 2.0A), a prioridade depende também do número da caixa de correio (*mailbox*), em que a caixa de correio (*mailbox*) 15 é a de maior prioridade.

Já no caso do protocolo CAN estendido (versão 2.0B), a prioridade da mensagem também depende do valor gravado no conjunto de bits *Transmit Priority Level* (TPL) do registrador MSGCTRL. Caso duas *mailboxes* apresentem o mesmo valor em TPL, a prioridade é da *mailbox* de maior número.

2.5 Principais registradores do módulo eCAN utilizados na comunicação entre DSPs

O bom funcionamento de uma rede CAN permite uma boa comunicação entre os dispositivos. Para isso, o conhecimento dos registradores de controle e de configuração do protocolo eCAN é de suma importância, visto que é através deles que define-se o modo de operação e identifica-se os problemas que ocorrem em operação.

2.5.1 Registradores da Mailbox

- MSGID (*Message Identifier*): armazena os 15 bits (versão 2.0A) ou os 32 bits (versão 2.0B) dos bits de identificação;
- CANMDL e CANMDH (*Message Data Low and Data High*): cada um pode armazenar até 4 bytes de dados a serem enviados;
- MSGCTRL (*Message Control*): por meio desse registrador, pode-se definir o tamanho do dado a ser enviado e habilitar o modo requerimento de transmissão remota (*Remote Trasmission Request*) de uma caixa de correio de recepção. Para a versão CAN 2.0B (protocolo CAN estendido), pode-se determinar o nível da prioridade de transmissão para uma caixa de correio de transmissão;

2.5.2 Registradores Controle e Estado

- CANME (*Mailbox Enable*): registrador com função de habilitar ou desabilitar cada caixa de correio;
- CANMD (*Mailbox Direction*): pode-se definir se uma caixa de correio será de transmissão ou de recepção;

- CANTRS (*Transmission Request Set*): registrador responsável por informar ao controlador CAN que uma certa caixa de correio possui um dado a ser transmitido;
- CANTRR (*Transmission Request Reset*): registrador usado para cancelar uma requisição de transmissão (CANTRS) de uma caixa de correio, que não está sendo processada.
- CANTA (*Transmission Acknowledge*): informa se a mensagem de uma caixa de correio foi enviada com sucesso. É dever do programador reinicializar esse registrador, cada vez que atuar;
- CANAA (*Abort-Acknowlegde*): informa se houve falha no envio da mensagem de uma dada caixa de correio. É dever do programador reinicializar esse registrador, cada vez que atuar;
- CANRMP (*Received Message Pending*): informa que uma certa caixa de correio recebeu uma mensagem com êxito. É dever do programador reinicializar esse registrador, cada vez que atuar;
- CANMC (*Master Control*): responsável pela escolha da versão CAN a utilizar, pela habilitação de modos de operação como o modo autoteste (*Self Test Mode*), pela determinação da ordem de transferência dos dados entre outras atribuições. Alguns de seus bits precisam de permissão EALLOW;
- CANES (*Error and Status*): nesse registrador pode-se verificar o tipo de erro ocorrido, a transmissão e a receptação de mensagens;
- CANGIF0/1 (*Global Interrupt Flag 0/1*): informa que a condição para gerar uma certa interrupção ocorreu;
- CANTEC (*Transmit Error Counter*): contador de erros ocorridos na transmissão de mensagens. Segue as regras do confinamento de falha (*fault confinement*) expostas no anexo A;
- CANREC (*Receive Error Counter*): contador de erros ocorridos na receptação de mensagens. Segue as regras do confinamento de falha (*fault confinement*) expostas no anexo A;
- CANOPC (*Overwrite Protection Control*): registrador responsável por evitar que uma mensagem sobrescreva outra durante o processo de transmissão;
- CANTIOC (*TX I/O Control*): Configura o pino CANTX para uso do periférico eCAN;
- CANRIOC (*RX I/O Control*): Configura o pino CANRX para uso do periférico eCAN;

- CANBTC (*Bit Timing Configuration*): nesse registrador pode-se configurar a taxa de transmissão de bit (*baudrate*), a quantidade de segmento que o módulo eCAN possui para determinar o nível lógico de cada bit e o tempo de cada segmento;

2.6 Tipos de mensagem

Para o protocolo CAN, existem vários tipos de mensagens com estrutura próprias e funções distintas. Dependendo do objetivo na comunicação, cada tipo de mensagem se adequa melhor ao funcionamento da rede CAN.

Com o objetivo de mostrar suas características e seus propósitos na comunicação, os tipos de mensagens foram abordados.

2.6.1 Mensagem de dado (*Data message*)

Esse tipo de mensagem apresenta o formato de estrutura igual ao exposto na Figura 6 e tem o objetivo de enviar o dado de um parâmetro de um dispositivo para outro.

Como para cada parâmetro é destinado uma identificação (*identifier bits*), então, quando o dispositivo for transmitir uma mensagem, este informa a identificação do parâmetro no registrador MSGID de uma caixa de correio de transmissão. Em seguida, o dado a transmitir, é armazenado nos registradores MDL e MDH. Dessa forma, os dispositivos, que utilizam em sua lógica tal parâmetro, devem ter uma caixa de correio de receptação com a mesma identificação registrada em seu respectivo MSGID.

Para esse tipo de mensagem, os bits RTR da caixa de correio de transmissão e de receptação devem armazenar o valor 0, informando que essa caixa de correio só transmite e recebe dados.

2.6.2 Mensagem de requisição (*Request message*)

Como o nome já diz, é uma mensagem de requisição de dado com objetivo de solicitar o dado de um parâmetro a um dispositivo.

Diferente da mensagem de dado, será transmitido à rede uma estrutura remota (*remote frame*) o qual é idêntico a estrutura da mensagem de dado (*data frame*) da Figura 6, exceto pela ausência dos *bytes* de dados.

Nesse caso, o dispositivo interessado na obtenção de dados de um outro dispositivo, faz uso de caixas de correio de receptação configuradas para o envio de *remote frame* (RTR = 1) ao barramento CAN. O dispositivo que possuir uma caixa de correio de transmissão com identificador igual ao identificador do *remote frame*, poderá enviar, automaticamente, o dado armazenado nos seus registradores MDL e MDH à caixa de correio de receptação de origem.

2.6.3 Mensagem de resposta (*Reply message*)

Como o nome já informa, é uma mensagem de resposta enviada após ser solicitada por meio de uma mensagem de requisição.

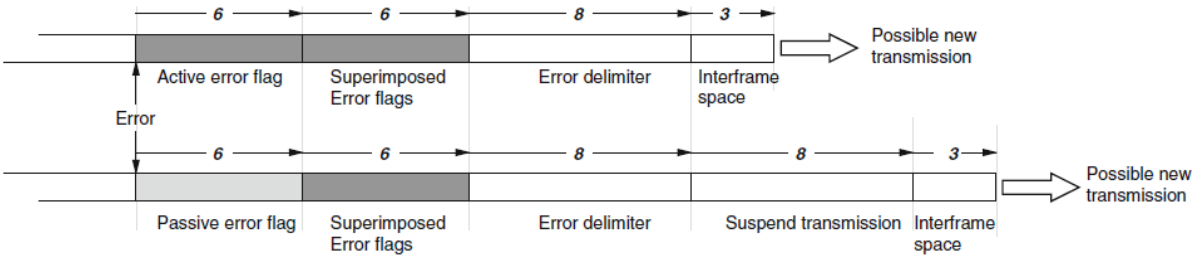
A estrutura da mensagem de resposta é igual à da mensagem de dado. O que muda é a forma como essa mensagem é enviada a mailbox de origem da mensagem de requisição. Ao receber uma mensagem de requisição, a caixa de correio de transmissão, configurada para atuar no modo auto resposta (AAM = 1), ativa o correspondente bit TRS automaticamente para o envio do dado armazenado em seus registradores MDL e MDH. Após o envio, o bit TRS retorna ao estado lógico 0 automaticamente.

2.6.4 Mensagem de erro (*Error message*)

A mensagem de erro (*error message*) não segue a estrutura exposta na Figura 6, pois não é verdadeiramente uma estrutura. Na verdade, é o resultado da atuação sequencial de mecanismos de detecção de erro, de autodiagnóstico e de sinalização de erros. Além disso, são usadas medidas para confinamento de falhas (*fault confinement*), impedindo que o status da rede seja abalado por dispositivos defeituosos.

A sequência da sinalização de erro consiste na superposição de bandeiras de erro (*error flags*), transmitidas por diferentes dispositivos, possivelmente em momentos distintos, e seguidos por um campo delimitador de erro (*error delimiter field*). A figura 8 mostra como o ocorre essa sinalização.

Figura 8 – Sequência de bits após uma detecção de erro.



Fonte: DI NATALE (2012).

Segundo o protocolo CAN, são definidos cinco tipos de erros detectados e estão listados abaixo.

- Erro de bit (*Bit error)*: todo dispositivo apto a transmitir uma mensagem possui a capacidade de verificar, no momento da transmissão, o estado lógico no barramento CAN e comparar com o dado enviado. Caso sejam diferentes, o erro de bit é detectado. Deve-se considerar que no momento do processo de arbitragem, essa verificação não é considerada verdadeira;
- Erro de coisa (*Stuff error)*: ocorre sempre que um sexto bit recessivo consecutivo é detectado na mensagem, informando que a regra do *bit stuff* não foi seguida corretamente;
- Erro CRC (*CRC error*): no dispositivo transmissor, um polinômio é gerado pela concatenação dos bits do campo de arbitragem, do campo de controle, do campo de dados e de 15 bits do campo CRC em estado dominante. Esse polinômio gerado é o dividendo que será dividido pelo polinômio gerador expresso na equação (1). O resto dessa divisão será o código CRC a ser enviado durante a transmissão da mensagem. Já no dispositivo de recepção, os dados serão lidos e outro polinômio dividendo será gerado formado pelos mesmos campos do anterior. Entretanto, o campo CRC armazena o resto da primeira divisão. Assim, esse novo polinômio é dividido novamente pelo polinômio gerador. Se o resto dessa nova divisão for uma sequência de 15 bits dominantes, significa que a mensagem foi transmitida e recebida com sucesso. Caso contrário, houve um erro na transmissão e o erro CRC será gerado;

$$X^{15} + X^{14} + X^{10} + X^{8} + X^{7} + X^{4} + X^{3} + 1 \qquad (1)$$

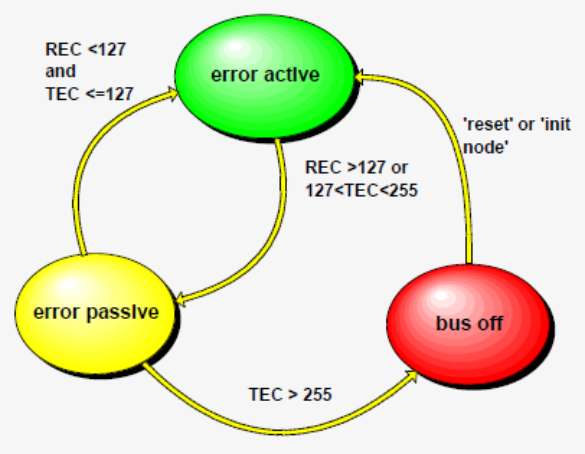
- Erro de formato (*Form error*): existem alguns bits que só podem apresentar um estado lógico. Por exemplo o bit delimitador do CRC, o bit delimitador do ACK, os bits do EOF e os bits do IFS. Todos esses citados só podem apresentar estado lógico recessivo, mas quando algum desses apresenta estado lógico dominante, ocorre o erro de formato ocorre quando há alguma violação no formato da mensagem, por exemplo, a falta da transmissão de um bit do campo CRC, entre outros;
- Erro de reconhecimento (*Acknowledgement error*): ao transmitir uma mensagem, o controlador CAN do transmissor determina que o bit ACK é recessivo. Quando o controlador CAN do receptor recebe a mensagem, este determina que o bit ACK seja dominante. Assim , essa mudança de estado do sinal funciona como a confirmação que a mensagem foi recebida. No caso de não haver essa mudança de estado lógico no bit ACK, ocorre um erro de reconhecimento;

Para auxiliar no diagnóstico do erro, existem 3 estados (figura 9) em que o gerenciamento de erro, de cada dispositivo, se baseia. Dependendo do estado, uma diferente estrutura de mensagem de erro (*error frame*) estará presente no barramento CAN. Os estados são denominados como erro ativo (*active error*), erro passivo (*passive error*) e barramento desativado (*bus off*). Esses três estados possuem o objetivo de identificar potenciais dispositivos defeituosos, para assim isolá-los e analisá-los sem que a rede CAN pare de funcionar.

- Estado de erro ativo (*Error active state*): com os contadores TEC ou REC apresentando valores entre 0 e 127, funciona normalmente, enviando e recebendo mensagens. Se um erro ocorrer, uma mensagem de erro ativo será transmitida à rede;
- Estado de erro passivo (*Error passive state*): com os contadores TEC ou REC apresentando valores entre 128 e 256, o envio e recepção de mensagens continua, mas caso ocorra algum erro, uma mensagem de erro passivo será transmitida;
- Barramento desativado (*Bus off*): com valores armazenados no contador TEC maiores que 256, o dispositivo é isolado do barramento CAN, não permitindo a transmissão nem o recebimento de mensagens. Nesse caso, também é

impossibilitado de enviar qualquer mensagem de erro antes de ser reinicializado;

Figura 9 – Esquemático da dinâmica dos estados.



Fonte: FRATICELLI (2013).

.

A mudança entre os estados se baseia nos valores dos dois contadores, TEC e REC. Segundo o *fault confinement*, diferentes valores são somados ou subtraídos a esses contadores, dependendo do sucesso ou do fracasso da ação de enviar e de receber as mensagens. As regas de atuação do *fault confinement* são abordadas no ANEXO A.

A qualquer momento, os dispositivos podem estar em estado de erro ativo (*active error*) ou de erro passivo (*Passive Error*). Se o dispositivo detecta um erro estando no estado de erro ativo, este transmite um aviso (*active error flag*) que consiste na sequência de seis bits dominantes. Em contra partida, se o dispositivo está no estado de erro passivo e um erro é detectado, um aviso (*passive error flag*) formado por seis bits recessivos consecutivos é transmitido.

Segundo a fabricante Bosch, quando os contadores de erro, TEC ou REC, atingem valores superiores a 96, a comunicação apresenta péssima qualidade, assim pode-se concluir que o barramento CAN apresenta problemas que devem ser averiguados para sua melhoria.

3 CODE GENERATION: COMANDOS E FUNÇÕES

Normalmente, em cursos e aplicações de engenharia, softwares de simulação estão muito presentes no cotidiano de estudantes, de pesquisadores e de profissionais devido ao seu grande poder de efetuar cálculos, de análise, de simulação de sistemas, de depuramento de códigos (*debug*) entre outros.

Habitualmente, mesmo programadores de microprocessadores experientes gastam um bom tempo no desenvolvimento de um programa escrito em uma linguagem estruturada como C por exemplo, devido a possíveis erros de implementação e configuração de registradores.

Com o objetivo de facilitar essa implementação, o pacote Embedded Coder® desenvolvido pela MathWorks® para a família C2000 da Texas Intruments™ foi desenvolvido para facilitar e agilizar a implementação por meio do *software* Simulink®. Este *software* permite a simulação e a análise de programas construídos por blocos de funções.

3.1 Comandos para instalação

Para poder usar a ferramenta Embedded Coder®, deve-se instalar alguns arquivos antes. Assim, ao executar o comando *checkeEnvSetup('ccsv5','f28335 eZdsp','check')* na tela de comando do MATLAB®, o próprio software irá informar quais arquivos devem ser instalados, como mostra a figura 10. (25)

Dessa maneira, pode-se verificar que, para esse exemplo, são requisitadas versões mínimas para o *software* Code Coposer Studio™, para o pacote Code Generation Tools, para o pacote DSP/BIOS, para o pacote XDC Tools e para pacotes Flash Tools. Nos próprios sites da Texas Instruments™ e da MathWorks® é possível encontrar esses softwares e pacotes.

Figura 10 - Informações mostradas com comando checkEnvSetup

```
1. CCSv5 (Code Composer Studio)
   Your version    : 6.1.0
   Required version: 5.0 or later
   Required for    : Code Generation
   TI_DIR="C:\ti\ccsv6"

2. CGT (Texas Instruments C2000 Code Generation Tools)
   Your version    : 6.0.2
   Required version: 5.2.1 to 6.0.2
   Required for    : Code generation
   C2000_CGT_INSTALLDIR="C:\Program Files (x86)\Texas Instruments\C2000 Code Generation Tools 6.0.2"

3. DSP/BIOS (Real Time Operating System)
   Your version    : 5.41.11.38
   Required version: 5.33.05 to 5.41.11.38
   Required for    : Code generation
   CCSV5_DSPBIOS_INSTALLDIR="C:\Program Files\Texas Instruments\bios_5_41_11_38"

4. XDC Tools (eXpress DSP Components)
   Your version    : 3.31.03.43.
   Required version: 3.16.02.32 or later
   Required for    : Code generation

5. Flash Tools (TMS320C28335 Flash APIs)
   Your version    : 2.10
   Required version: 2.10
   Required for    : Flash Programming
   FLASH_28335_API_INSTALLDIR="C:\ti\controlSUITE\libs\utilities\flash_api\2833x\28335\v210"
```

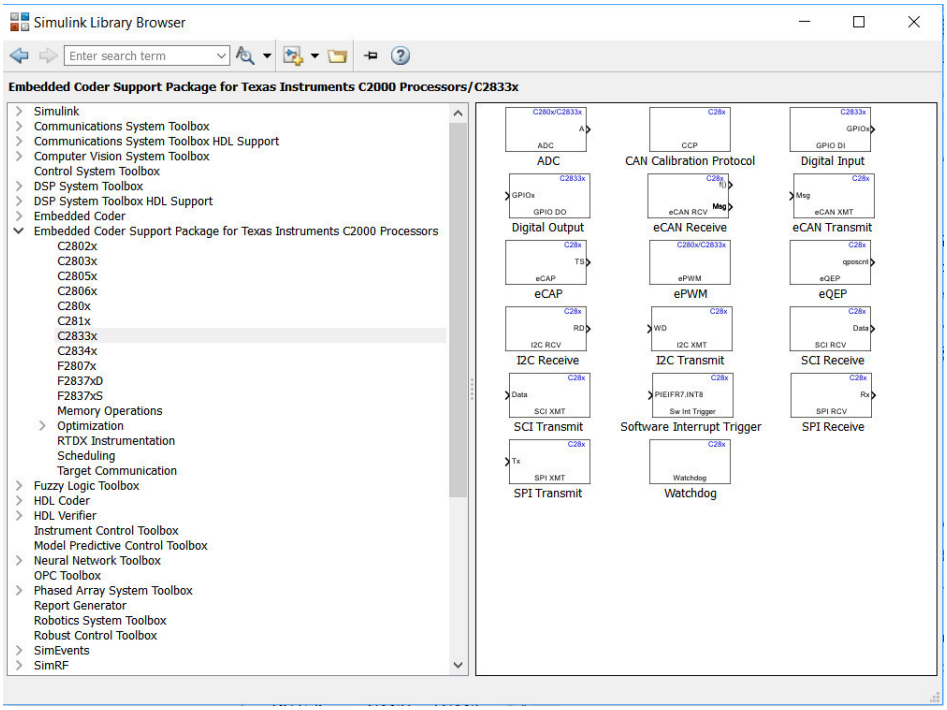
Fonte: MATLAB® (2019).

Para o auxílio na instalação de alguns softwares necessários para o bom funcionamento do Embedded Coder®, por exemplo Code Composer Studio™ e ControlSuite™, o comando supportPackageInstaller executa um pacote que auxilia o programador na instalação dos recursos.

Depois da instalação, o Navegador da Biblioteca do Simulink® (*Simulink Library Browser*) adquire uma nova biblioteca Embedded Coder destinada a família de processadores C2000 da Texas Instruments™, composta por blocos de funções referentes aos periféricos dos diversos processadores dessa família. Na figura abaixo, pode-se ver os blocos de funções dos periféricos presentes em processadores F2833x. Alguns desses blocos serão usados na criação das simulações de teste do barramento CAN.

Além desses, essa nova biblioteca (figura 11) possui blocos específicos para controle discreto de motores (Transformação de Clarck e de Park, *Space Vector Generator*, Controlador PID, medição de velocidade por exemplo), para manejo de dados no formato IQmath, troca de dados em tempo real (*Real-Time Data Exchange* RTDX), Blocos de interrupção, entre outros.

Figura 11 - Simulink Library Browser com bibliotecas Embedded Coder para família de processadores C2000.

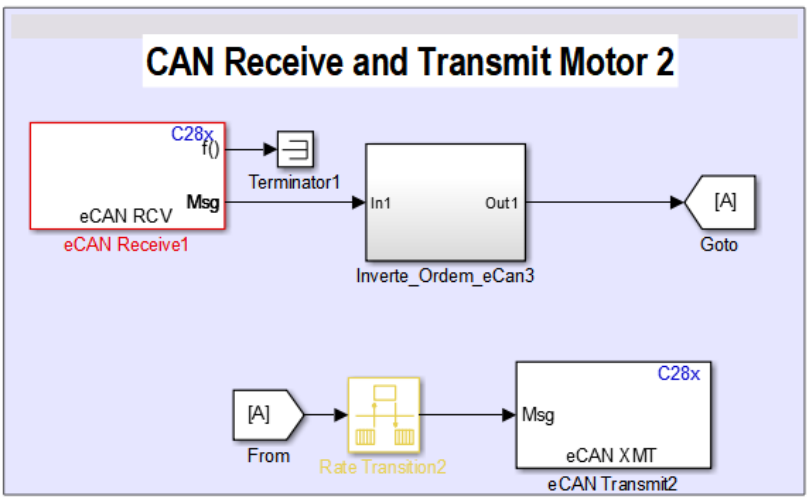


Fonte: Simulink® (2019).

3.2 Modelo no Simulink®

Um modelo Simulink®, Figura 12, foi desenvolvido de forma que, por meio deste, fosse possível receber e enviar dados a uma rede CAN, após ser executada a ferramenta Code Generation, ou seja, o modelo seria convertido em um arquivo em linguagem C e inserido em um DSP.

Figura 12 – Modelo Simulink® para envio e recepção de dados na rede CAN.



Fonte: Próprio autor.

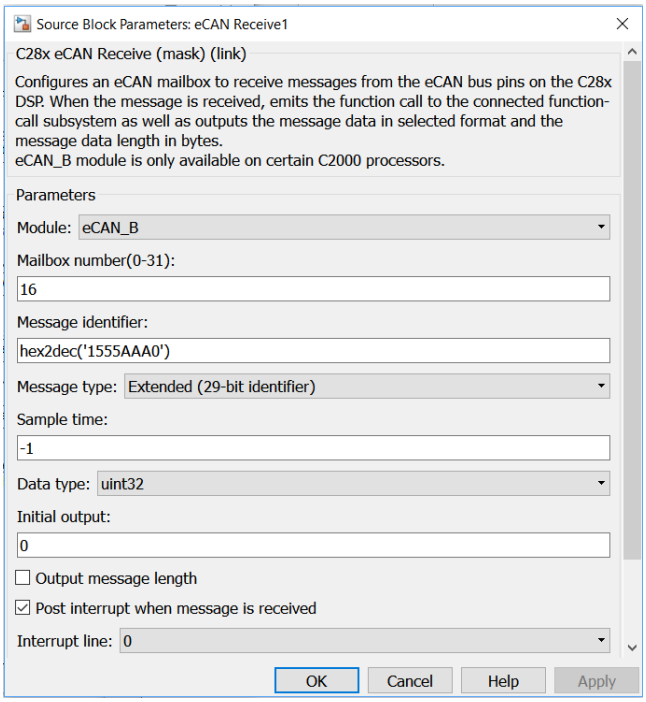
A ideia do teste era receber e transmitir a mesma mensagem da rede, utilizando blocos como recepção eCAN (*eCAN Receive*), transmissão eCAN (*eCAN Transmit*), taxa de transmissão (*Rate Transition*) além dos blocos de operação de bit (*Bit Operations*).

3.3 Blocos de Funções

O bloco recepção CAN (figura 13) não utiliza os nomes dos registradores do DSP para indicar que tipo e valor de deve ser inserido. Assim, pode-se verificar o seguinte:

- Em *Module*, pode ser escolhido o canal eCAN do DSP a ser utilizado no momento;
- Especifica-se qual mailbox está em uso para recepção;
- Em *Message Identifier*, deve-se inserir os 29bits ou 15bits com o valor do identificador;
- A versão do protocolo, 2.0A (identificador de 15bits) ou 2.0B (identificador de 29bits), é definida em *Message Type*;
- O tempo de amostragem (transmissão de toda a mensagem), em segundos, é determinado em *Sample Time*. O valor -1 indica que o tempo de amostragem desse bloco é o mesmo inerente ao sistema ou ao bloco conectado à montante;
- A mensagem recebida será adaptada ao formato estipulado em *Data Type*;
- Ao iniciar a execução do código, o canal apresentará o valor escrito em *Initial output*;
- Marcando o espaço *Output Message Length*, o tamanho da mensagem recebida estará informado em um pino de saída;
- Marcando o espaço *Post Interrupt*, indica à CPU que, após o recebimento de uma mensagem, um gatilho para geração da interrupção de recepção por um canal eCAN escolhido em *Module*, será gerado. Se a interrupção estiver ativa, será gerada uma interrupção;
- Cada canal eCAN possui duas linhas de interrupção, *ECAN0INT* e *ECAN1INT*, que podem ser escolhidas em *Interrupt Line*;

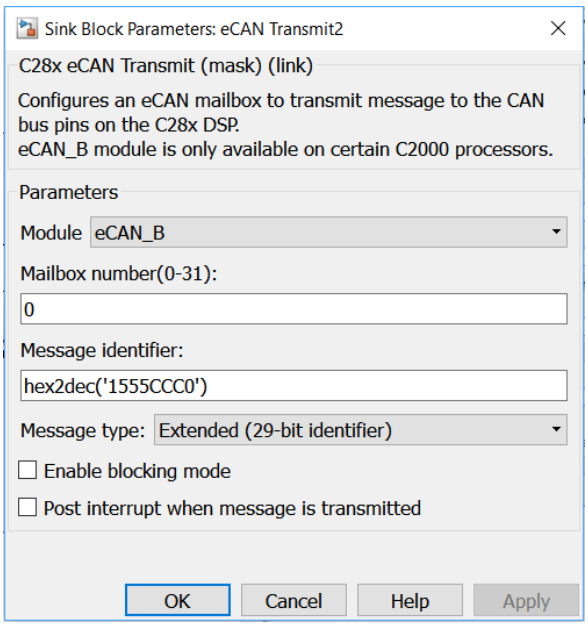
Figura 13 – Janela de configuração do bloco eCAN Receiver.



Fonte: Simulink® (2019).

A configuração do bloco *eCAN Transmit* (figura 14), responsável pela transmissão de uma nova mensagem, é uma versão simplificada da configuração do *eCAN Receiver*. Diferencia-se pela função *Enable Blocking Mode* que determina se o bloco eCAN esperará, indefinidamente ou não, por um *Transmit Acknowledge*.

Figura 14 – Janela de configuração do bloco eCAN Transmit.

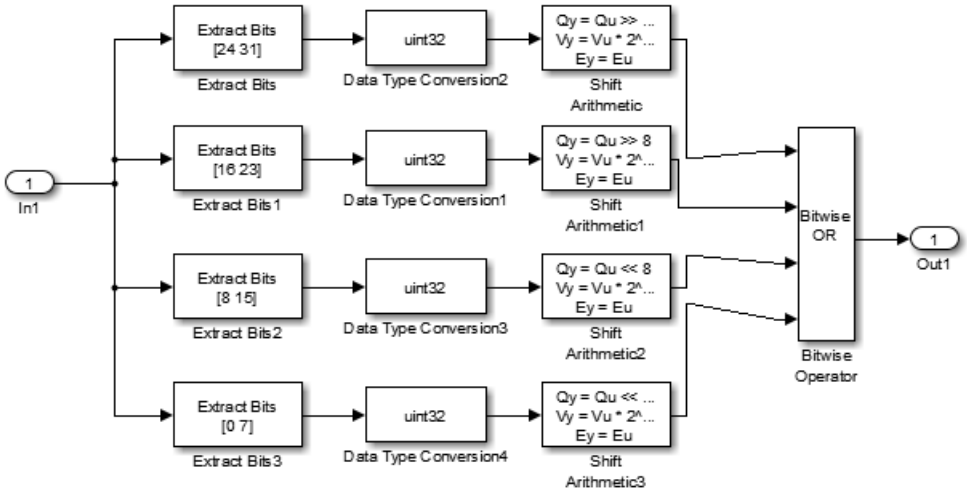


Fonte: Próprio autor.

Uma particularidade desses blocos (*eCAN Transmit* e *eCAN Receive*) é o envio, byte a byte, de uma mensagem, em que o byte mais significativo transmitido de um outro dispositivo se torna o byte menos significativo no dispositivo de recepção.

Para corrigir esse problema, um bloco de inversão dos bytes da mensagem recebida (Inverte_Ordem_eCan) foi criado e está mostrado na figura 15.

Figura 15 - Bloco de Inversão de mensagem.



Fonte: Próprio autor.

No bloco Inverte_Ordem_eCan, a entrada recebia dados de 32 bits. Em seguida, o bloco *Extract Bits* era usado para selecionar o respectivo byte de interesse no formato de 8 bits. Depois, através do bloco de função *Data Type Conversion*, o formato de 8bits era mudado para 32 bits, em que os bits à esquerda acrescentados apresentavam valores nulos. Na sequência, esse byte era realizado operações de deslocamento de bits com o intuito de reorganizar os bytes na posição correta. Por fim, todos as quatro partes sofriam a ação do operador lógico OU (OR) ao passarem pelo bloco Bitwise. Na saída, o dado de 32 bits reorganizado era obtido.

A ferramenta Code Generation™ simplifica a programação de placas de desenvolvimento dedicando menos tempo a implementação do código em linguagem C. Entretanto, há uma limitação para o programador quanto análise de erros, sendo necessário verificar sua operação no CCS.

4 RESULTADOS

Normalmente, na indústria, os braços robóticos utilizados em linhas de produção, seguem trajetórias pré-definidas para realizar suas tarefas. Assim, todos os graus de liberdade podem se movimentar em conjunto, otimizando o tempo gasto em cada trajetória.

Dessa forma, para o braço robótico em estudo, a criação de um meio de comunicação entre as placas de desenvolvimento eZdsp F28335 é de grande importância, visto que tem seus graus de liberdade controlados por esses dispositivos. Assim, a implementação desse meio, possibilita que o movimento dos graus de liberdade do braço seja realizado simultaneamente.

Consequentemente, para esse trabalho, o principal objetivo era estabelecer um meio de comunicação entre os graus de liberdade de um braço robótico, em que faça uso de dos periféricos já existentes, possibilitando o controle independente do módulo de comunicação, mas atuariam em conjunto. No princípio, os testes para seguir a trajetória somente eram realizados em um módulo por vez, visto que não havia a troca de dados entre esses módulos de controles.

Nesse capítulo são apresentadas a sequência de formação do meio de comunicação entre as três placas de desenvolvimento eZdsp F28335, através do Code Composer Studio™ (CCS) em linguagem C para microprocessados e por meio do Simulink® em funções de blocos.

Inicialmente, foi realizada a comunicação tendo os eZdsps programados via CCS. Depois os códigos foram adaptados para a linguagem de blocos.

4.1 Braço Robótico

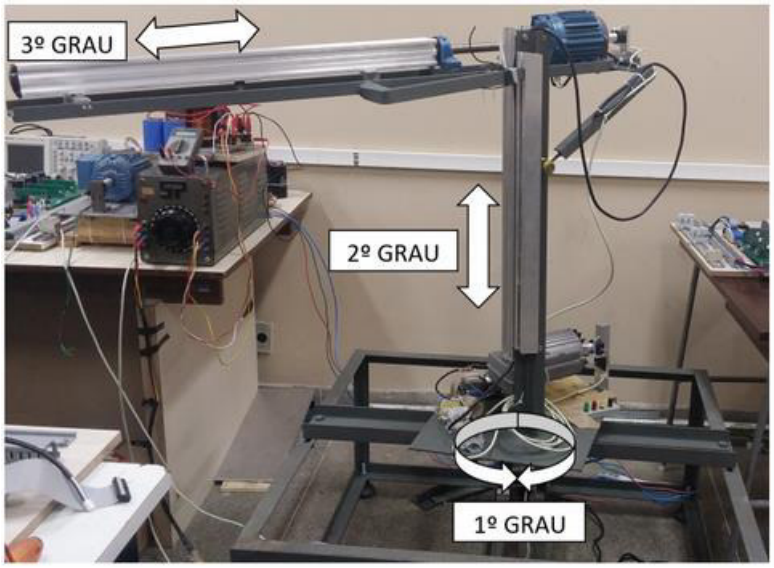
O braço robótico (figura 16) situado no laboratório Grupo de Pesquisa de Automação e Robótica (GPAR), pertencente ao Departamento de Engenharia Elétrica (DEE) na Universidade Federal do Ceará (UFC).

Esse braço foi projetado para acomodar cinco graus de liberdade. O primeiro grau comporta a base do manipulador, o segundo, do movimento vertical, o terceiro, do movimento horizontal e os dois restantes referentes a garra. Atualmente, a garra não está instalada, então somente os três primeiros graus de liberdade estão disponíveis. O primeiro grau apresenta

movimento rotacional na base do manipulador, o segundo, apresenta movimento linear na vertical e o terceiro, um movimento linear na horizontal.

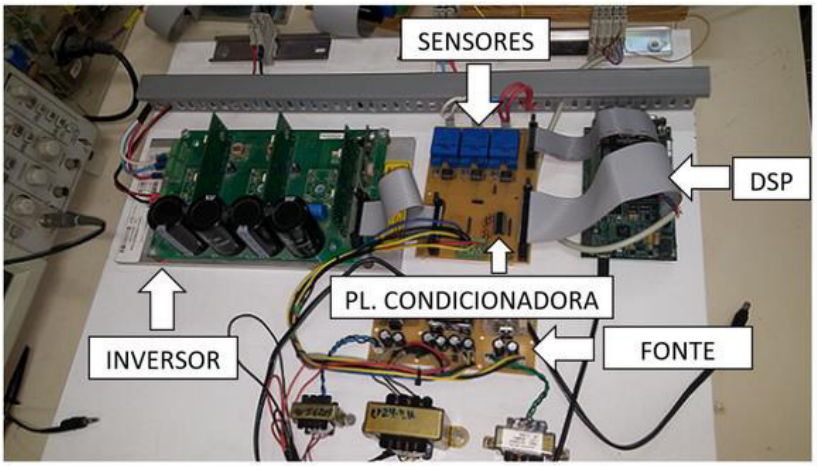
Cada grau de liberdade é comandado por um módulo de controle, figura 17, constituído por um inversor da Semikron®, modelo SKS 18G B6111 V12, uma placa de desenvolvimento eZdsp F28335, uma placa com os sensores de corrente, um *encoder* incremental e uma placa de condicionamento.

Figura 16 – Braço robótico.



Fonte:REBOUÇAS (2017).

Figura 17 – Módulo de controle de cada grau de liberdade.



Fonte: REBOUÇAS (2017).

4.2 Dispositivo de controle – eZdsp F28335

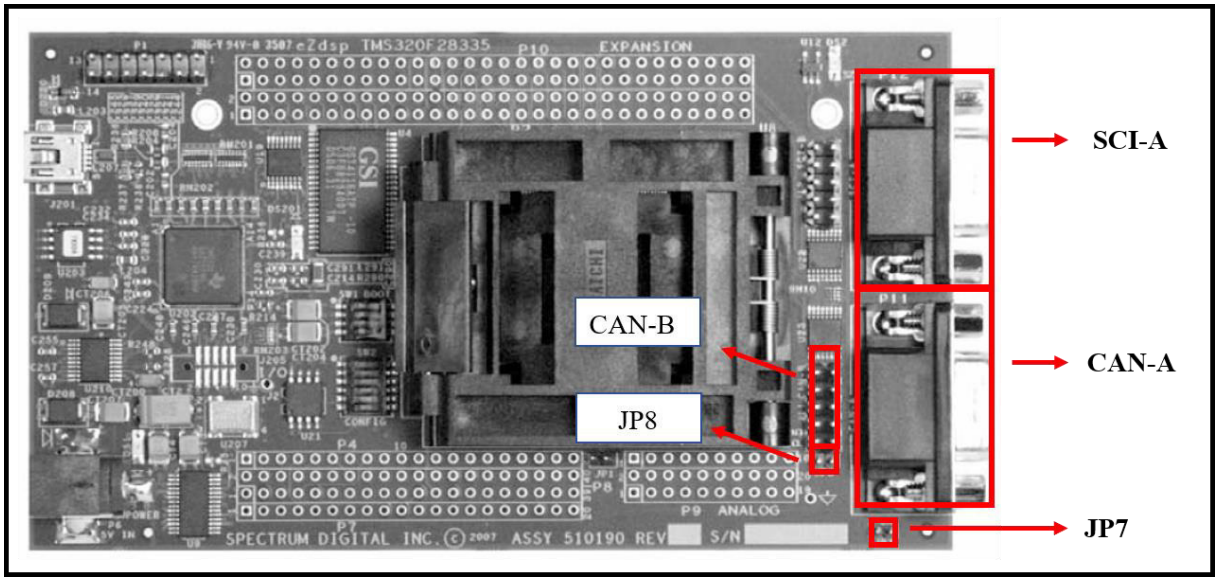
Desenvolvida pela National Instruments™, a placa de desenvolvimento eZdsp F28335 faz uso do circuito integrado do microcontrolador TMS320F28335. Possui periféricos de comunicação SPI, SCI, I2C e eCAN além do gerador PWM, conversores AD, interpretador de *encoder* entre outros.

Como o foco principal desse trabalho é a comunicação pelo protocolo eCAN, então serão abordadas suas particularidades, pinos de conexão, etc.

O circuito integrado TMS320F28335, disponibiliza de dois canais independentes eCAN, o CANA e o CANB, onde estão disponíveis 32 *mailboxes* por canal a depender da versão de protocolo CAN selecionada. Além disso, para cada canal, esse CI já possui uma unidade controladora CAN.

No intuito de conectar diretamente ao barramento CAN, a placa de desenvolvimento eZdsp F28335 (figura 18) já possui um transceptor CAN SN65HVD235 para cada canal e seu respectivo resistor terminal. Assim, basta conectar os pinos da placa ao barramento. As pinagens de cada canal CAN estão ilustradas na figura 19.

Figura 18 – Placa de desenvolvimento eZdsp F28335.

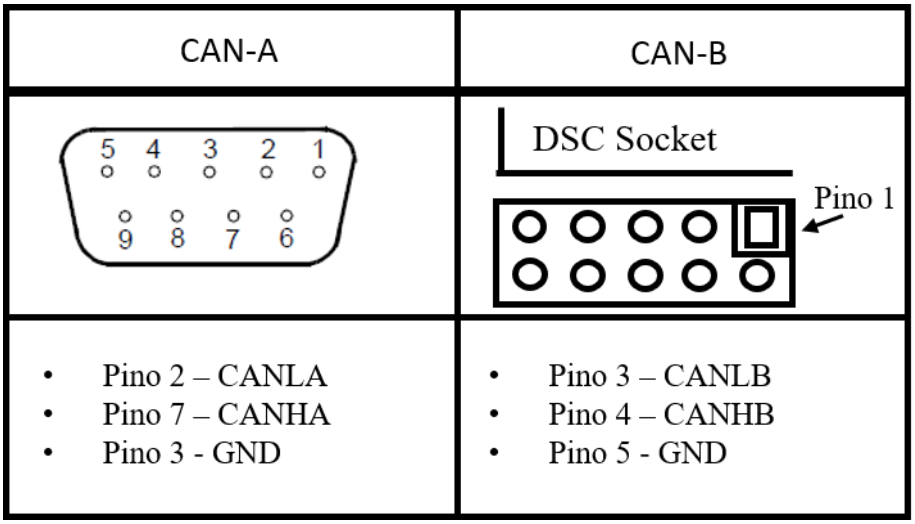


Fonte: Technical reference Texas Instruments™.

A placa eZdsp F28335 já possui seus próprios resistores terminais de 120 Ω para cada canal, denominados JP7 e JP8 referentes aos canais CAN-A e CAN-B respectivamente. Caso o canal seja o dispositivo mais próximo da extremidade do barramento, então o conector

JPx deve estar estabelecendo curto-circuito nesses dois pinos. Caso contrário, esse conector deve ser retirado. (21)

Figura 19 – Pinagem dos canais CAN.



Fonte: Próprio autor.

4.3 Rede com três eZdsp F28335 programados via CCSv6

Primeiramente, para saber se os canais estavam funcionando corretamente, cada canal foi testado usando o código exemplo fornecido pela Texas Instruments™ que fazia uso do modo autoteste (*Self-Test Mode*).

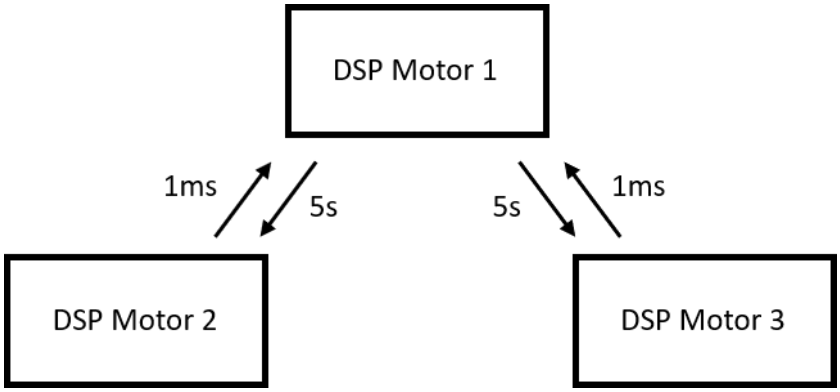
As três placas foram nomeadas como DSP Motor1, DSP Motor 2 e DSP Motor 3.

Após isso, com o intuito de estabelecer uma comunicação bidirecional entre as placas, tentou-se concretizar a comunicação entre duas placas. Nesse momento, primeiramente, os comandos de transmissão foram inseridos em um laço infinito, mas somente um dos canais enviava a mensagem devido a regra de prioridade relacionada ao valor do identificador de cada mensagem. Para tentar corrigir, esses comandos de transmissão foram inseridos em uma interrupção de transmissão do canal CAN-A, mas o problema persistiu. Por fim, baseado na premissa que para algumas técnicas de controle são necessárias taxas de transmissão constantes, os comandos foram inseridos em uma interrupção do *timer* 0, apresentando uma taxa de transmissão constante.

O esquema de comunicação entre os três DSPs via barramento CAN é expresso na figura 20. Foi escolhido esse formato, pois serviria de base para o próximo esquema, onde

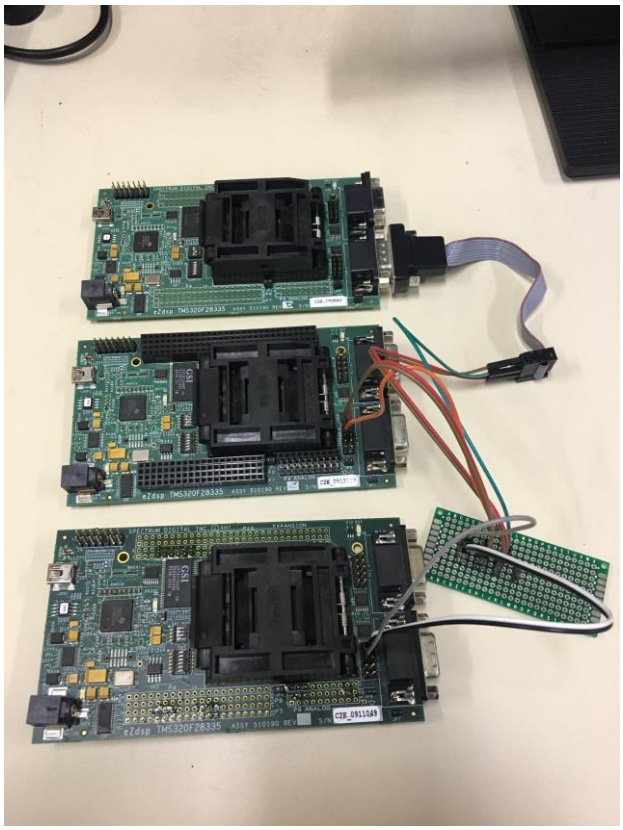
haveria a presença de um computador a enviar e a receber dados da rede CAN por meio do DSP Motor 1. Na figura 21, a conexão física entre os três DSPs é mostrada.

Figura 20 – Esquema da comunicação entre os três DSPs.



Fonte: Próprio autor.

Figura 21 – Conexão entre placas eZdsp F28335.

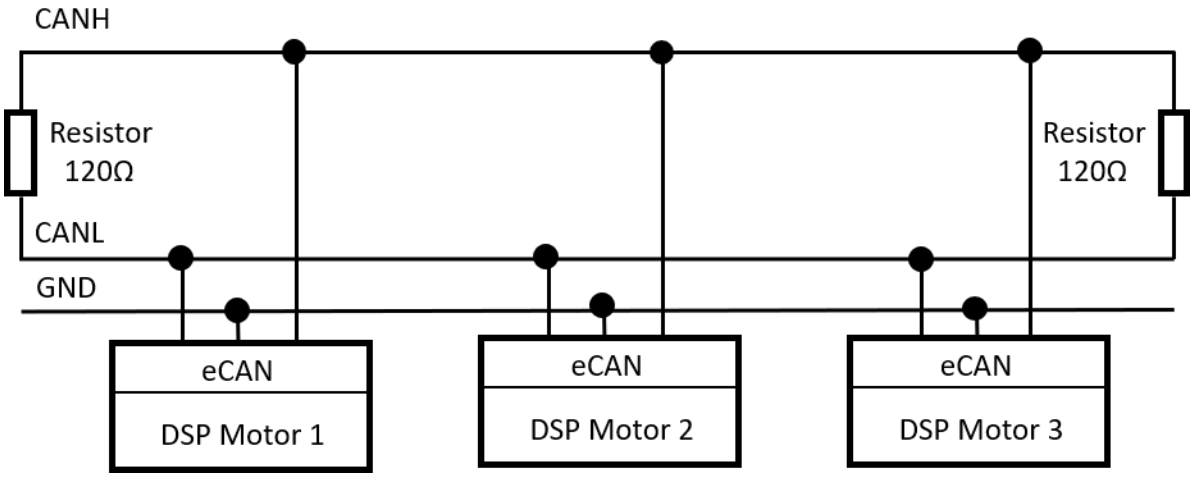


Fonte: Próprio autor.

Observando que a rede CAN do braço robótico não possui uma grande quantidade de parâmetros a serem transmitidos na rede, não haverá problemas quanto ao gerenciamento do trânsito de mensagens na rede e da utilização das *mailboxes* no envio ou na recepção de

vários dados de parâmetros. Assim, para verificar a comunicação entre os DSPs, foi utilizado uma quantidade reduzida de *mailboxes* de cada DSP.

Figura 22 – Estrutura da rede CANbus.



Fonte: Próprio autor.

Primeiramente, via programação em linguagem C, foi estabelecida a comunicação entre três placas eZdsp F28335 via barramento CAN, em que todas as placas eram programadas e depuradas (*Debug*) através do software Code Composer Studio™.

Em seguida, todas as mailboxes 0 foram configuradas para serem de transmissão. Já as *mailboxes* 16 dos DSP Motor 2 e DSP Motor 3, foram configuradas como de recepção para armazenar dados oriundos do DSP Motor 1. No DSP Motor 1, as mailboxes 16 e 17 foram configuradas como recepção para receber dados oriundos dos DSPs Motor 1 e Motor 2. Foi preciso configurar as *mailboxes* de cada canal para poderem transmitir e receber mensagens vinculadas a terceira placa.

Para fixar a taxa de transmissão dos dados, fez-se uso de interrupções do *Timer 0*, em que uma interrupção é gerada a cada 5s no DSP Motor 1 e a cada 1ms nos DSPs Motor 2 e 3. Dessa maneira, cada vez que a interrupção do timer 0 do DSP Motor 1 acontecer, um dado hexadecimal de até 8 bytes é transmitido à rede, destinado às *mailboxes* 16 dos DSPs Motor 2 e 3. Estes DSPs leem seus dados e reenviam, durante a interrupção do *Timer 0*, para as *mailboxes* 16 e 17 dos respectivos DSP Motor 2 e 3. Esses valores de taxas de transmissão foram escolhidos somente com o objetivo de poder visualizar o envio das mensagens.

Já para as *mailboxes* de recepção, a cada mensagem recebida, uma interrupção assíncrona entra em atividade para que seja feita a leitura de todas as mailboxes de recepção habilitadas.

Nesse esquema de comunicação, foi possível estabelecer, de forma satisfatória, a comunicação bidirecional para cada DSP, ou seja, cada DSP transmitia e enviava dados para a rede CAN. Nesse experimento, foram usados os tipos de mensagens *Data*, *Request* e *Reply message*.

4.4 Rede com três eZdsp F28335 programados via Simulink®

A rede apresentaria a mesma funcionalidade do experimento anterior, mas com a diferença de alguns DSPs serem programados por *softwares* distintos, seriam usadas as mesmas *mailboxes* e suas configurações.

A princípio, nesse experimento, um DSP era programado em linguagem C via CCS, enquanto os demais, por meio da ferramenta *Code Generator*, eram programados via Simulink®, utilizando blocos de funções do pacote Embedded Coder®.

Assim, como o Simulink® não possui uma ferramenta para depurar (*Debug*) o funcionamento do DSP programado via Simulink®, deve-se utilizar a janela *CCS Debug* no CCS para realizar a depuração do código oriundo do Simulink®.

A comunicação entre os DSPs na rede foi bem-sucedida, verificando-se um fluxo bidirecional de dados em todos os dispositivos.

Um padrão de envio e de recepção de dados pode ser verificado. Os dados de 8 *bytes* eram enviados separadamente, em pacotes de 1 *byte*, em que o *byte* menos significativo enviado pelo DSP programado em linguagem C seria o *byte* mais significativo recebido pelo DSP programado via Simulink®. Dessa forma, para se obter o real dado enviado, a ordem dos *bytes* deveria ser reorganizada no Simulink®.

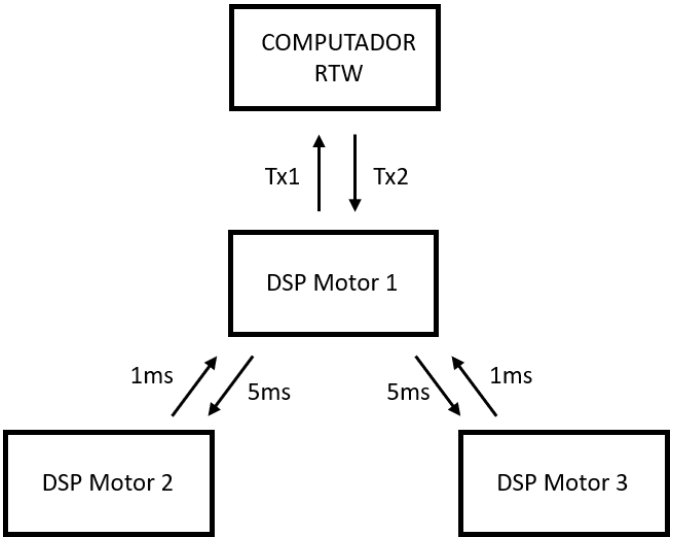
4.5 Rede com três eZdsp F28335 programados via Simulink® e RTW

Pensando em utilizar o PC em conjunto com a rede CAN dos DSPs, pretendia-se realizar a implementação da ferramenta *Real-Time Workshop*® (RTW), desenvolvida pela MathWorks®, para possibilitar o desenvolvimento de projetos utilizando o conceito *Hardware in the Loop* (HIL).

Nesse experimento, a rede CAN apresentaria as mesmas características das anteriores, havendo o acréscimo das configurações referentes a ferramenta RTW.

O esquema dessa comunicação está exposto na figura 23 abaixo, em que a comunicação entre os DSPs se daria por meio de protocolo CAN e a comunicação entre DSP Motor 1 e o computador, por meio do protocolo SCI (mais conhecido como UART).

Figura 23 – Esquema de comunicação contendo o computador.



Fonte: Próprio autor.

A configuração do *software* MATLAB/Simulink® para operar com a ferramenta RTW está conforme o manual RTW [28].

Inicialmente, para teste as taxas de transmissão Tx1 e Tx2 assumiram valores de teste, por exemplo 1s.

Após enviar os respectivos códigos via Simulink® aos DSPs, foi realizada a configuração e montagem do código em blocos de função que usaria a ferramenta RTW no computador. Entretanto, ao tentar executar a simulação em Simulink® com RTW, infelizmente, o computador reiniciava. Vendo esse problema, fiz a mesma configuração em um notebook, mas ao tentar executar a mesma simulação, o sistema operacional do notebook travava.

Esse problema não foi resolvido, devido a limitações da configuração dos computadores utilizados.

CONCLUSÃO

Utilizar protocolo CAN para estabelecer um meio de comunicação entre os DSPs, foi de excelente proveito para o projeto do braço robótico, pois é ideal para transferir dados de pequeno tamanho como valores de corrente e de tensão elétrica, valores de posição e torque entre outros.

Como, primeiramente, foi realizada a rede CAN entre somente dois DSPs, pode-se verificar a facilidade de adicionar um outro dispositivo a rede CAN.

As taxas de transmissão, de 1 ms e de 5 s, usadas devem ser readequadas a uma aplicação em tempo real.

As ferramentas para detectar a ocorrência de erros são muito eficazes e poderosas, pois indicam todos os estados que a rede CAN pode apresentar, facilitando na busca da causa de cada problema que ocorreu.

O protocolo eCAN disponibilizado no DSP, como está normalizado somente pelas duas normas ISO 11898 e CAN Specification 2.0, faz uso somente das duas primeiras camadas, camada física e camada de ligação. Dessa forma, fica a cargo do programador elaborar um algoritmo de gerenciamento da transmissão dos dados. Esse gerenciamento é mais requisitado quando a quantidade de dados que devem ser enviados é maior que a quantidade de caixas de correios (*mailboxes*) disponíveis no canal CAN do DSP.

A ferramenta *Code Generator* ajuda o programador a gastar menos tempo no desenvolvimento de códigos a serem executados no DSP. Entretanto, para depurar a execução do código no DSP, é melhor fazê-lo através do CCS *Debug*, pois este apresenta todo o aparato para monitoramento dos registradores.

Como trabalho futuro, pode-se configurar e utilizar a ferramenta RTW para efetuar simulações em tempo real segundo o conceito *Hardware in the Loop*, onde o processamento do computador pode ser usado em conjunto dos DSPs.

Fica a proposta para utilizar placas de desenvolvimento, como ESP32 e CAN32, para realizar aplicações de IoT com o braço robótico.

Pode ser feito também uma rede CAN mais complexa ou multiredes CAN em que permitam a comunicação entre os dispositivos, os sensores e os equipamentos do braço robótico.

REFERÊNCIAS

1 ROBERT BOSCH GMBH. **CAN Specification Version 2.0**. Stuttgart, 1991.

2 COPPERHILL TECHNOLOGIES. **A Brief Introduction to the SAE J1939**. Disponível em : <https://copperhilltech.com/a-brief-introduction-to-the-sae-j1939-protocol/>. Acesso em 13 mai. 2019.

3 NAVET, Nicolas; SIMONOT-LION, Françoise. **Automotive Embedded Systems Handbook**. Estados Unidos: CRC Press, 2008.

4 TIANMING, Liu. **CAN Bus Application Layer Protocol Design for Space Manipulator Communication System**. IEEE 2nd ADVANCED INFORMATION TECHLONOGY, ELECTRONIC AND AUTOMATION CONTROL CONFERENCE (IAEAC), 2008, China.

5 RENESAS ELECTRONIC AMERICA INC. **Using CAN bus Serial Communications in Space Flight Applications**. Tokio, 2018.

6 CCS ELECTRONICS. **J1939 Explained – A Simple Intro (2019)**. Disponível em : < https://www.csselectronics.com/screen/page/simple-intro-j1939-explained/language/en#J1939-Intro-Dummies-Basics> Acessado em: 22 mai. 2019.

7 KINO, Satoshi. **Application and Benefits of an Open DeviceNet Control System in the Forest Products Industry**. ANNUAL PULP AND PAPER INDUSTRY TECHNICAL CONFERENCE, 1999, Seattle, Estados Unidos.

8 CCS ELECTRONICS. **CANopen Analyser – Easily Log Data from your Machines**. Disponível em: < https://www.csselectronics.com/screen/page/canopen-data-logger>. Acessado em: 22 mai. 2019.

9 DEXTRA. **Internet das coisas: entenda como impacta as empresas**. Disponível em: <https://dextra.com.br/pt/iot-internet-das-coisas/>. Acessado em: 24 mai. 2019.

10 SCHLEGEL, Christian. **The role of CAN in the age of Ethernet and IoT.** 16th INTERNATION CAN CONFERENCE (ICC), 2017, Nuremberg, Germany.

11 ESPRESSIF SYSTEMS. **Controller Area Network (CAN)**. Disponível em: < https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/can.html> Acessado em: 24 mai. 2019.

12 KOYANAGI, Fernando. **Protocolo CAN – Yes, we can!**. Disponível em: < https://www.fernandok.com/2018/07/protocolo-can-yes-we-can.html>. Acessado em: 24 mai. 2019.

13 ALLAN, Alasdair. **The CAN32 – an ESP32 Based CAN Bus Board**. Disponível em : <https://blog.hackster.io/the-can32-an-esp32-based-can-bus-board-af5c9e1bd8ec>. Acessado em: 24 mai. 2019.

14 TEXAS INSTRUMENTS. **TMS320x2833x, 2823x Serial Peripheral Interface (SPI)**.

Dallas, 2009.

15 TEXAS INSTRUMENTS. **TMS320F2833x, TMS320F2823x Digital Signal Controllers (DSCs)**. Dallas, 2019.

16 ARROW ELECTRONICS. **SPI vs I2C Protocols – Pros and Cons**. Disponível em: < https://www.arrow.com/en/research-and-events/articles/spi-vs-i2c-protocols-pros-and-cons>. Acessado em: 21 mai. 2019.

17 TEXAS INSTRUMENTS. **TMS320x2833x, 2823x Inter-Integrated Circuit (I2C) Module**. Dallas, 2009.

18 TEXAS INSTRUMENTS. **TMS320x2833x, 2823x Serial Communication Interface (SCI)**. Dallas, 2009.

19 TEXAS INSTRUMENTS. **TMS320x2833x, 2823x Enhanced Controller Area Network (eCAN)**. Dallas, 2009.

20 JAVA T POINT. **OSI Model**. Disponível em : < https://www.javatpoint.com/osi-model>. Acessado em: 27 mai. 2019.

21 SPECTRUM DIGITAL INCORPORATED. **eZdsp F28335 – Technical Reference**. Estados Unidos, 2007.

22 TEXAS INSTRUMENTS. **SN65HVD23x 3.3V CAN Bus Tranceivers**. Dallas, 2019.

23 DI NATALE, Marco et al. **Understanding and Using the Controller Area Network Communication Protocol**. Estados Unidos, Springer, 2012.

24 BORMANN, Frank. **Texas Instruments Teaching Materials – F2833x Controller Area Network**. Estados Unidos, Data de fabricação desconhecida.

25 ELRAJOUBI, Akrem; ANG, Simon S; ABUSHAIBA, Ali. **TMS320F28335 DSP Programming using MATLAB SimulinkEmbedded Coder :Techniques and Advancements.** Workshop on Control and Modeling for Power Electronics (COMPEL), 2017, Stanford, Estados Unidos.

26 FRATICELLI, Jose Carlos Molina. **Simulink Code Generation: Tutorial for Generating C Code from Simulink Models using Simulink Coder**. Estados Unidos, BiblioGov, 2013.

27 BARRIENTOS, Cristian Erardo Miranda. **Programación de DSP a través de Herramientas Disponibles em MATLAB 7.0/Simulink**. 2011. Monografia (Graduação em Engenharia Elétrica) – Universidad de Magallanes, Punta Arenas, 2011.

28 MATHWORKS. **Simulink Desktop Real-Time – User's Guide**. Natick, Estados Unidos, 2015.

29 MATHWORKS. **CCS Support by MathWorks Release**. Disponível em: <http://software-dl.ti.com/ccs/esd/documents/ccs_matlab.html>. Acessado em:01 jun.

2019.

30 TEXAS INSTRUMENTS. **Controller Area Network Physical Layer Requirements**. Dallas, 2008.

31 JIANG, Benoni. **Linhas de transmissão – Sinais transmitidos e refletidos.** Disponível em: <https://www.youtube.com/watch?v=ozeYaikI11g> . Acessado em:29 jun. 2019.

32 SOMASHEKHARA, G.B.. **CAN Protocol basics. Part4, Bus Termination.** Disponível em :< https://www.youtube.com/watch?v=nGpYRT0nnvE&list= PLqU5orWr2scAuIs5 _uwk6o2cUm4Q0r> . Acessado em: 29 jun. 2019;

33 REBOUÇAS, Lucas R. **Controle de um motor de indução trifásico servo posicionador aplicado a um manipulador robótico utilizando controle de campo orientado.** 2017. 116f. Dissertação (Mestrado em Engenharia Elétrica) – Departamento de Engenharia Elétrica, Universidade Federal do Ceará, Fortaleza, 2017;

ANEXO A – Regras do Fault Confinement

O *fault confinement* se baseia nos estados *error active*, *error passive* e *bus off* em que um dispositivo deve estar situado em algum desses três estados.

As características e funcionalidades de um dispositivo situado em cada estado é expressa abaixo:

- Um dispositivo em *error active* não apresenta limitações quanto às suas funcionalidades na comunicação CANbus. Nesse estado só pode transmitir *active error flags* quando for detectado um erro;
- Um dispositivo em *error passive* é impossibilitado de enviar *error active flag*. Nesse estado, participa da rede CAN normalmente até a ocorrência de um erro. Desse modo, somente uma *error passive flag* pode ser transmitida. Como consequência, o dispositivo espera a próxima transmissão;
- Um dispositivo em *bus off*, é impossibilitado de afetar a comunicação via CAN. Como consequência, o módulo periférico CAN do dispositivo é desativado resultando em uma desconexão do dispositivo à rede CAN.

Essa ferramenta *fault continement*, faz uso de contadores, *transmit error count (TEC)* e *receive error count (REC)*, para determinar em qual estado o dispositivo se situa. Esses contadores seguem as regras estipuladas pela Bosch™:

Tabela – Regras de mudanças nos valores dos contadores REC e TEC.

Dispositivo como receptor	Mudança no contador	Quando o dispositivo
Receive error count (REC)	+1	Detecta um erro exceto um *bit error* durante um *active error flag* ou *overload flag*.
	+8	Detecta um bit dominante como o primeiro bit após enviar uma *error flag*.
	+8	Detecta um *bit error* durante o envio de um *active error flag* ou *overload flag*.
	-1	Recebe com êxito uma mensagem e o contador REC≤127.
	Qualquer n pertencente ao intervalo 119≤n≤127	Recebe com êxito uma mensagem e o contador REC>127.
	+8	Detecta mais de 7 bits dominantes consecutivos após o envio de uma *active error flag*, *passive error flag* ou *overload error flag* (+8 para cada 8 bits dominantes

		adicionais).
Dispositivo como transmissor	Mudança no contador	Quando o dispositivo
Transmit error count (TEC)	+8	Envia um error flag
	+8	Detecta um bit error durante o envio de uma *active error flag* ou de uma *overload flag*.
	-1	Transmite com êxito uma mensagem.
	+8	Detecta mais de 7 bits dominantes consecutivos após o envio de uma *active error flag*, *passive error flag* ou *overload error flag* (+8 para cada 8 bits dominantes adicionais).

Fonte: DI NATALE, 2012.

ANEXO B – Versões compatíveis do CCS e do MATLAB

A empresa MathWorks® desenvolve pacotes para a utilização de versões MATLAB®, Simulink® e Embedded Coder® em união a versões do CCS. Duas ferramentas são usadas:

- *ert.tlc*: usada para gerar um código em linguagem C/C++ a partir de um modelo Simulink e inseri-lo diretamente no DSP via Code Composer Studio™. Essa ferramenta suporta desde a versão CCSv3 em diante para famílias C2000 e C6000, entre outros dispositivos;
- *idelink_ert.tlc*: criada primeiramente para a versão CCSv3.3, atende todas as necessidades dessa versão, mas, para as versões CCSv4 e CCSv5, existem funções que não estão inclusas, assim resultando em um suporte parcial;

Tabela – Versões compatíveis entre MATLAB® e Code Coposer Studio™.

MATLAB VERSION	CCS VERSION SUPPORTED	PRODUCT OR PACKAGE REQUIRED
MATLAB 7.9 (R2009b)	CCSv3	Target Support Package Embedded IDE Link (product)
MATLAB 7.10 (R2010a)		
MATLAB 7.11 (R2010b)	CCSv3 and CCSv4	
MATLAB 7.12 (R2011a)	CCSv3 and CCSv4	Embedded Coder (product, includes Embedded IDE Link and Target Support Package) Support from 64-bit MATLAB
MATLAB 7.13 (R2011b)		
MATLAB 7.14 (R2012a)		
MATLAB 8.0 (R2012b)	CCSv3 up to CCSv5	
MATLAB 8.1 (R2013a)		
MATLAB 8.2 (R2013b)	CCSv3 up to CCSv5	Embedded Coder Free downloadable Support Package for C2000
MATLAB 8.3 (R2014a)		
MATLAB 8.4 (R2014b)		
MATLAB 8.5 (R2015a)	CCSv3 up to CCSv6	
MATLAB 8.6 (R2015b)		
MATLAB 9.0 (R2016a)		
MATLAB 9.1 (R2016b)		
MATLAB 9.2 (R2017a)	CCSv3 up to CCSv6	CCS Project (CCSv5, CCSv6) generated when using TI C2000 Support Package Support for TI C2000 with CCSv3 discontinued (idelink_ert.tlc)
MATLAB 9.3 (R2017b)	CCSv3 up to CCSv7	Started support for C2000Ware (2017b)
MATLAB 9.4 (R2018a)		
MATLAB 9.5 (R2018b)	CCSv3 up to CCSv8	

Fonte : MATHWORKS® (2019)