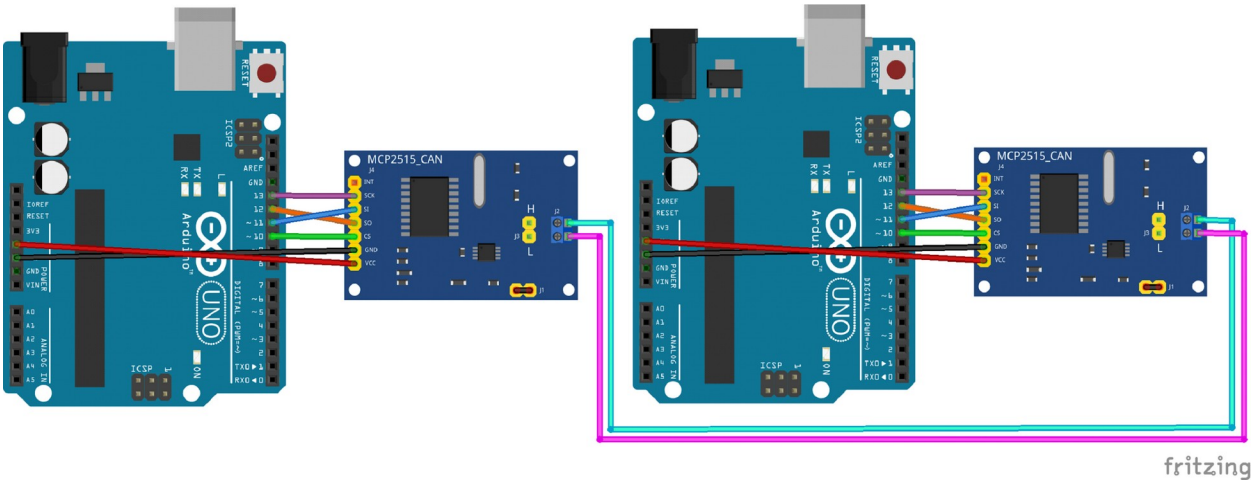
Barramento CAN Entre Arduinos UNO

Nesse artigo quero mostrar uma aplicação simples de implementar, e que pode esclarecer os conceitos que estão por trás de uma rede CAN. Utilizei dois Arduinos UNO e dois Controladores CAN MCP2515. Como o Microcontrolador que vem na placa Arduino - Atmega328p - não tem um periférico controlador CAN, então, com essa configuração podemos facilmente criar um barramento CAN.

Antes de apresentar os detalhes dessa aplicação é importante entender o básico sobre a CAN, sendo assim, com a aplicação funcionando podemos experimentar e consolidar os conceitos vistos neste artigo. Também esse artigo foi inspirado numa aplicação similar que eu vi na internet, link https://www.electronicshub.org/arduino-mcp2515-can-bus-tutorial/. Como pode ser constatado que não é um plágio, mas, algo relacionado, porém com abordagem diferente e na língua portuguesa para ajudar brasileiros que não conhecem outra língua.

Figura 1 - Configuração criada com o software fritzing (site http://fritzing.org)



Definições de Termos Utilizados

MCU - Microcontrolador
SPI - Interface Periférico Serial (Serial Peripheral Interface)
ISO - Organização Internacional de Padronização
OSI - Sistema de Interconecção Aberto (Open Systems Interconnection)
CAN - Rede Controladora de Área (Controller Area Network)
CANH - Condutor elétrico do barramento CAN (CAN High), onde os dados trafegam
CANL - Condutor elétrico do barramento CAN (CAN Low), onde os dados trafegam
Bus - Barramento, onde os dados trafegam
Frame - Pacote de bits

Bit Recessivo - Nível lógico '1' na entrada do Transceiver, mas, no barramento CAN é a diferença de potencial entre CANH e CANL, quando essa diferença for < 0,5 Volts.
Bit Dominante - Nível lógico '0' na entrada do Transceiver, mas, no barramento CAN é a diferença de potencial entre CANH e CANL. quando essa diferença for > 1,0 Volts.
CSMA/CD + AMP - Protocolo de comunicação com Senso-Portador e Acesso-Múltiplo, com Detecção de Colisão e Arbitragem em Prioridade de Mensagem (Carrier-Sense, Multiple-Access protocol with Collision Detection and Arbitration on Message Priority)

Introdução Sobre a CAN
A CAN foi desenvolvida pela BOSCH e disponibilizada em meados dos anos 80. Sua aplicação inicial foi realizada em ônibus e caminhões. Atualmente, é utilizada na indústria, em veículos automotivos, navios, tratores e outras aplicações. A CAN utiliza um sistema de transmissão de mensagens multi-mestre, onde todos os módulos podem se tornar mestre em determinado momento e escravo em outro. Suas mensagens são enviadas em regime multicast, caracterizada pelo envio à todos os módulos existentes na rede.

A CAN pode se comunicar numa taxa máxima de até um Megabit por segundo (Mbps). Diferente de uma rede de comunicação tradicional, como USB ou Ethernet, a CAN não envia grandes frames de dados ponto-a-ponto, isto é, de um determinado nó para outro nó, sob a supervisão de um mestre. A CAN envia muitas mensagens curtas, como por exemplo, temperatura ou RPM. É uma rede de comunicação serial assíncrona. Na comunicação assíncrono o transmissor transmite o sinal de sincronismo para a sincronização do receptor, marcando o início de transmissão do frame.

Outro conceito bastante interessante é o NRZ (Non Return to Zero), onde cada bit (0 ou 1) é transmitido por um valor de tensão específico e constante. Na CAN todos os nós são conectados em pontos comuns (CANH e CANL). Segundo os padrões a CAN alta velocidade, comunica numa taxa de 512 Kbps (ISO 11898-2), e a CAN baixa velocidade, comunica numa taxa de 125 Kbps (ISO 11898-3).
A velocidade de transmissão dos dados é inversamente proporcional ao comprimento do barramento. A maior taxa de transmissão especificada é de 1Mbps considerando-se um barramento de 40 metros.

Fatores relevantes da CAN:

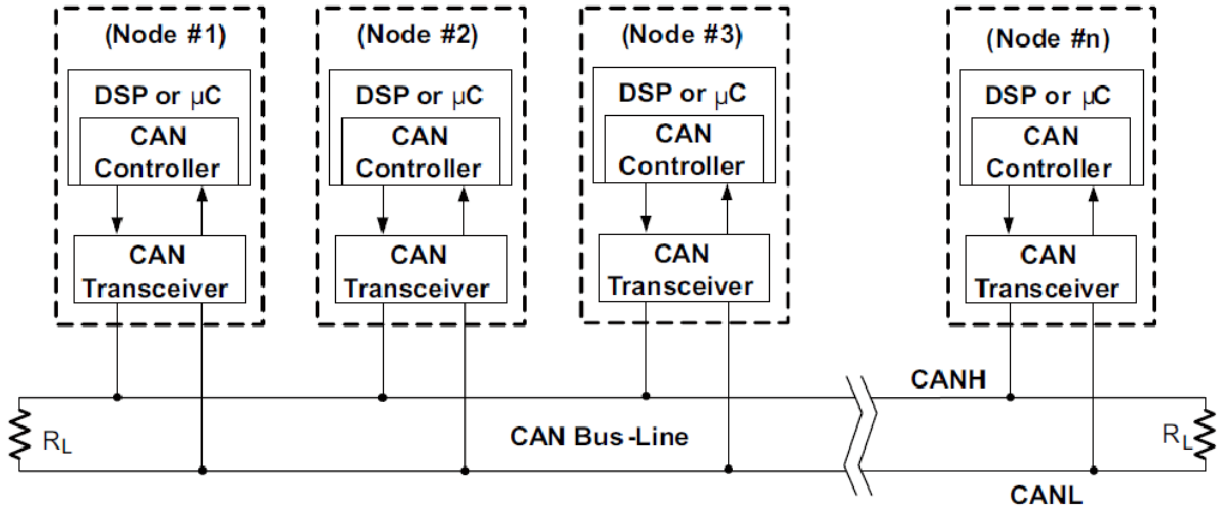
• Baixo custo
• Rápida (até 1 Mbps)
• Alta imunidade a interferências
• Robusta em detecção e tratamento de erros (confiabilidade)
• Novos nós são facilmente adicionados

O Padrão da CAN

A CAN é um barramento de comunicação serial definido pela ISO desenvolvido originalmente para a indústria automotiva para substituir o chicote de fiação complexo por um barramento de dois fios. A especificação exige alta imunidade a interferências elétricas e a capacidade de auto-diagnosticar e reparar erros de dados. Esses recursos levaram à popularidade da CAN em vários setores.

O Barramento Físico

Figura 2 - Visão geral do barramento CAN. Extraída de um artigo técnico no site da Texas Instruments http://www.ti.com/lit/an/sloa101b/sloa101b.pdf



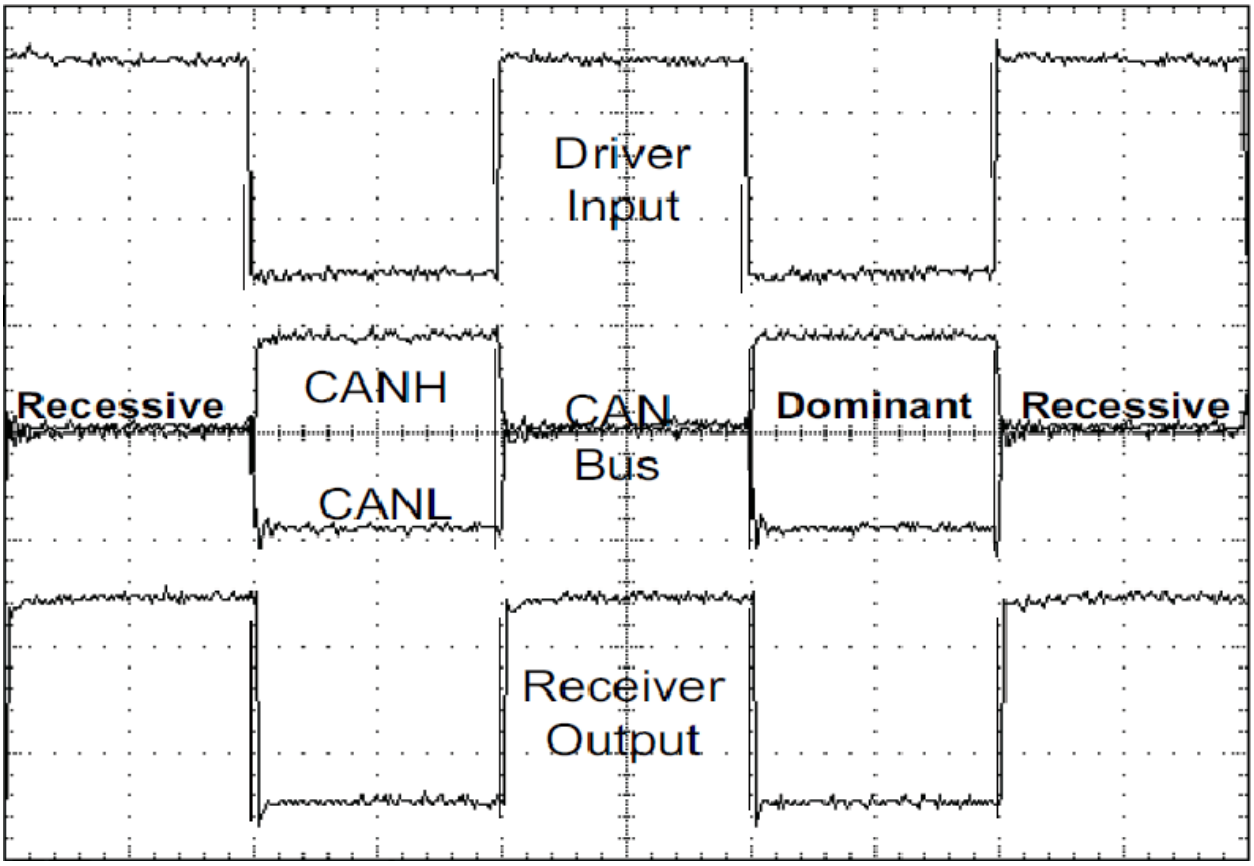
O barramento CAN é formado por um cabo de par trançado. Se considerarmos fios elétricos como o meio de transmissão dos dados, existem duas formas de se constituir um barramento CAN, dependentes diretamente da quantidade de fios utilizada. Existem barramentos baseados em 2 e 4 fios. Os barramentos com 2 e 4 fios trabalham com os sinais de dados CANH e CANL. No caso dos barramentos com 4 fios, além dos sinais de dados, um fio com o VCC (alimentação) e outro com o

GND (referência) fazem parte do barramento, levando a alimentação às duas terminações ativas da rede.

Considerando o barramento de 2 e 4 fios, seus condutores elétricos devem ser trançados e não blindados. Os dados enviados através da rede devem ser interpretados pela análise da diferença de potencial entre os fios CANH e CANL, por isso, o barramento CAN é classificado como par trançado diferencial. Este conceito atenua fortemente os efeitos causados por interferências eletromagnéticas, uma vez que qualquer ação sobre um dos fios será sentida também pelo outro, causando flutuação em ambos os sinais para o mesmo sentido e com a mesma intensidade. Como o que vale para os módulos que recebem as mensagens é a diferença de potencial entre os condutores CANH e CANL, e esta permanecerá inalterada, a comunicação não é prejudicada.

No Barramento CAN, os dados não são representados por bits em nível "0" ou nível "1". São representados por bits Dominantes e bits Recessivos, criados em função da condição presente nos fios CANH e CANL. A figura a seguir ilustra os níveis de tensão em uma rede CAN, assim como os bits Dominantes e Recessivos.

Figura 3 - Tela de um Osciloscópio ajustado em 2 Volts por divisão. Extraída de um artigo técnico no site da Texas Instruments
http://www.ti.com/lit/an/sloa101b/sloa101b.pdf



O sinal do Driver de entrada (Driver Input) e do Driver de saída (Receiver Output) são os sinais que os MCU processam, então, são as entradas e saídas do Transceiver, portanto, não são propriamente os sinais do barramento CAN. Olhe novamente a figura 2, então, entenderá o que está sendo frisado nesse paragrafo.

A CAN Padrão e a CAN Estendida

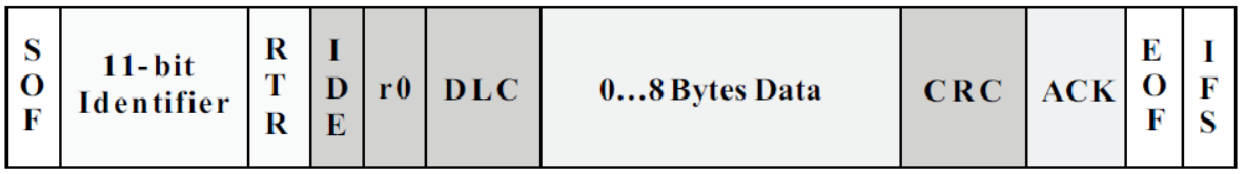
A comunicação CAN utiliza um protocolo com senso-portador, acesso-múltiplo, com detecção de colisão e arbitragem em prioridade de mensagem (CSMA/CD + AMP). CSMA significa que cada nó deve monitorar o barramento e aguardar um período de inatividade antes de enviar uma mensagem e também durante um período de inatividade do barramento todos os nós têm igual oportunidade para transmitir uma mensagem. CD+AMP significa que se dois nós transmitir ao mesmo tempo a colisão ocorre. As mensagens são mantidas intactas mesmo colidindo. Toda mensagem que

perdeu na arbitragem é automaticamente re-transmitida no próximo período de inatividade do barramento. As colisões são resolvidas por meio de uma arbitragem bit a bit, com base em uma prioridade no campo identificador de uma mensagem. O identificador de prioridade mais alta sempre obtém acesso ao barramento; ou seja, o último sinal lógico alto no identificador continua transmitindo porque é a prioridade mais alta. Como todos os nós de um barramento participam da escrita de todos os bits "conforme estão sendo escritos", um nó de arbitragem sabe se colocou o bit com lógica alta no barramento.

O padrão ISO-11898: 2003, com o identificador padrão de 11 bits, fornece taxas de comunicação de 125 kbps a 1 Mbps. O padrão foi posteriormente alterado com o identificador "estendido" de 29 bits. O campo identificador padrão de 11 bits fornece identificadores de mensagem de 211 ou 2048 diferentes, enquanto o identificador estendido de 29 bits fornece 229 ou 537 milhões de identificadores diferentes.

A CAN Padrão

Figura 4 - Frame da mensagem padrão. Extraída de um artigo técnico no site da Texas Instruments http://www.ti.com/lit/an/sloa101b/sloa101b.pdf



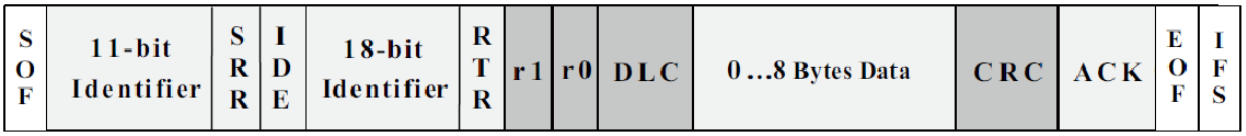
O significado dos campos de bit da CAN padrão são:

• SOF - O bit único dominante de Início De Frame (Start Of Frame - SOF) marca o início de uma mensagem, e seu bordo de descida é usado para sincronizar os nós em um barramento após ficar inativo.
• Identificador - O identificador da CAN padrão, de 11 bits de comprimento, estabelece a prioridade da mensagem. Quanto menor o valor binário, maior sua prioridade.
• RTR - O bit único de Solicitação de Transmissão Remota (Remote Transmission Request - RTR) é dominante quando informação é requerida de outro nó. Todos os nós recebem a solicitação, mas o identificador determina o nó especificado. Os dados de resposta também são recebidos por todos os nós e utilizados por qualquer nó interessado. Dessa maneira, todos os dados usados em um sistema são uniformes.

• IDE - Um bit único dominante de IDentificador Estendido (IDentifier Extension - IDE) significa que um identificador CAN padrão sem extensão está sendo transmitido.
• r0 - Bit reservado (para possível uso em futuras alterações padrão).
• DLC - O Código de Comprimento de Dados (Data Length Code - DLC) de 4 bits contém o número de bytes de dados que estão sendo transmitidos, podendo variar de 0 a 8 bytes.
• Dados - Até 64 bits de dados de aplicação podem ser transmitidos, isto é, até 8 bytes.
• CRC - A Verificação de Redundância Cíclica (Cyclic Redundancy Check - CRC) de 16 bits, 15 bits mais um bit delimitador, contém a soma de verificação (número de bits transmitidos) dos dados da aplicação anterior para detecção de erros. O bit delimitador é recessivo e marca o final do CRC.
• ACK - Cada nó que recebe uma mensagem precisa sobrescreve esse bit recessivo na mensagem original com um bit dominante, indicando que uma mensagem sem erros foi enviada. Se um nó receptor detectar um erro e deixar esse bit recessivo, ele descarta a mensagem e o nó emissor repetirá a mensagem após re-arbitragem. Dessa maneira, cada nó reconhece (Acknowledge - ACK) a integridade de seus dados. O reconhecimento utiliza 2 bits, um é o bit de Reconhecimento (ACK) e o segundo é um delimitador. O bit delimitador é recessivo e marca o final do ACK.
• EOF - Este campo de 7 bits de Fim De Frame (End Of Frame - EOF) marca o final de uma mensagem, e desativa o processamento de bits, indicando um erro de preenchimento quando dominante. Quando 5 bits do mesmo nível lógico ocorrem sucessivamente durante a operação normal, um pouco do nível lógico oposto é inserido nos dados.
• IFS - Este Espaço Inter-Frame (InterFrame Space - IFS), de 7 bits, contém o tempo necessário para o controlador mover um frame recebido corretamente para sua posição correta em uma área de armazenamento de mensagem.

A CAN Estendida

Figura 5 - Frame da mensagem estendida. Extraída de um artigo técnico no site da Texas Instruments http://www.ti.com/lit/an/sloa101b/sloa101b.pdf



A mensagem CAN estendida é a mesma que a mensagem padrão com a adição de:

• SRR - O bit de Solicitação Remota Substituta (Substitute Remote Request - SRR) substitui o bit RTR na posição de mensagem padrão como um espaço reservado no formato estendido.
• IDE - Um bit recessivo no IDentificador Estendido (IDE) indica que mais bits identificador seguem. A extensão de 18 bits segue o IDE.
• r1 - Após os bits RTR e r0, um bit de reserva adicional foi incluído antes do bit DLC.

Uma Mensagem CAN

ARBITRAGEM

Uma característica fundamental da CAN é o estado lógico oposto entre o barramento e a saída do receptor conectada a entrada do driver. Normalmente, uma lógica alta é associada a um nível '1', e uma lógica baixa é associada a um nível '0'; mas não em um barramento CAN. É por isso que os transceptores CAN têm internamente os pinos do driver de entrada e saída passivamente puxado-cima (pull-up), de modo que, na ausência de qualquer entrada, o dispositivo automaticamente assume um estado de barramento recessivo em todos os pinos de entrada e saída.

O acesso ao barramento é orientado a eventos e ocorre aleatoriamente. Se dois nós tentarem ocupar o barramento simultaneamente, o acesso será implementado com uma arbitragem bit a bit não destrutiva. Não destrutivo significa que o nó vencedor da arbitragem continua com a mensagem, sem que a mensagem seja destruída ou corrompida por outro nó.

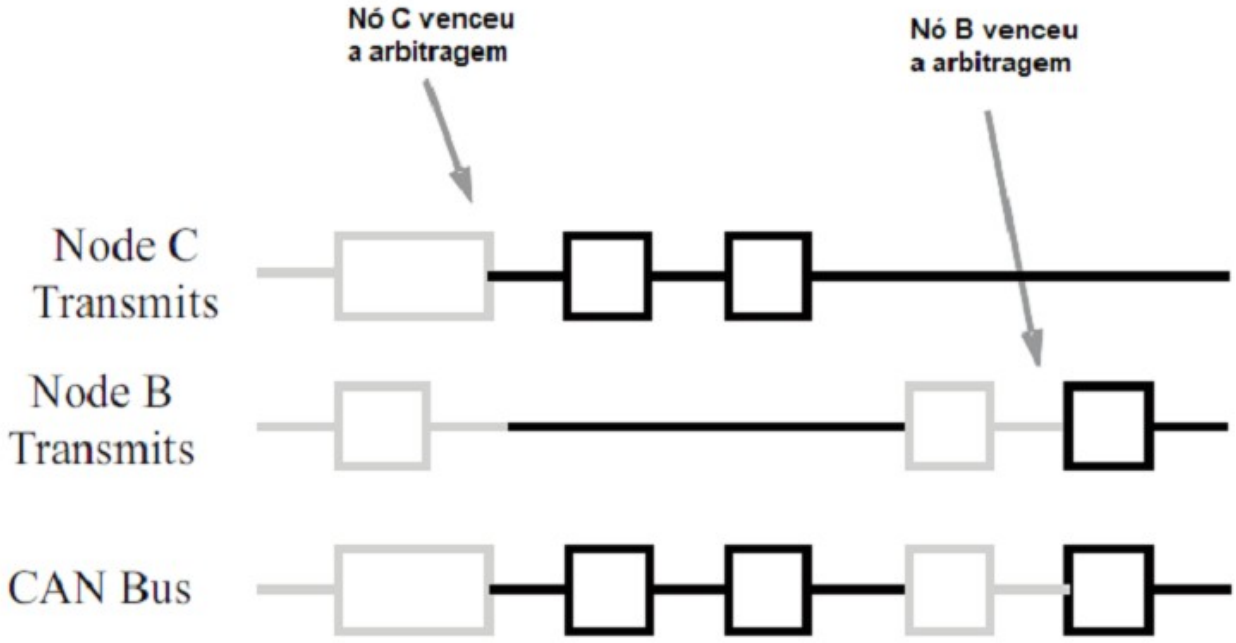
A atribuição de prioridade às mensagens no identificador é um recurso da CAN que a torna particularmente atraente para uso em um ambiente de controle em tempo real. Quanto menor o número binário do identificador da mensagem, maior a sua prioridade. Um identificador que consiste somente em zeros é a mensagem de prioridade mais alta em uma CAN, porque mantém o barramento dominante por mais tempo. Portanto, se dois nós começarem a transmitir simultaneamente, o nó que envia um último bit identificador como zero (dominante), enquanto o outro nó envia um (recessivo), detém o controle do barramento CAN e continua a completar sua mensagem. Um bit dominante sempre sobrescreve (vence) um bit recessivo em um barramento CAN.

Um nó que transmite constantemente monitora cada bit de sua própria transmissão. Esta é a razão, na qual os pinos de saída CANH e CANL do driver são

internamente ligados à entrada do receptor. O atraso de propagação do um sinal nesse loop interno é tipicamente usado como uma medida qualitativa de um transceptor CAN. Esse atraso de propagação é referido como o tempo de loop (tLOOP em uma folha de dados de um transceptor), mas assume uma nomenclatura variada dependendo do fornecedor.

O processo de arbitragem CAN é tratado por um controlador da CAN. Como cada nó monitora continuamente suas próprias transmissões, então, na figura a seguir, como o bit recessivo do nó B é sobrescrito pelo bit dominante de maior prioridade do nó C, B detecta antecipadamente que o estado do barramento não corresponde ao bit a ser transmitido, portanto, o nó B interrompe a transmissão enquanto o nó C continua com sua mensagem. Depois você pode ver outra tentativa de transmitir a mensagem é feita pelo nó B quando o barramento é liberado pelo nó C. Essa funcionalidade faz parte da camada de sinal físico da ISO 11898, o que significa que ela está contida inteiramente dentro do controlador da CAN e é completamente transparente para um usuário da CAN.

Figura 6 - Arbitragem de mensagens. Extraída de um artigo técnico no site da Texas Instruments http://www.ti.com/lit/an/sloa101b/sloa101b.pdf



A atribuição da prioridade da mensagem depende de um designer de sistema, mas os grupos da indústria concordam mutuamente sobre o significado e peso de certas mensagens. Por exemplo, um fabricante de acionamentos de motor pode

especificar que a mensagem 0010 é um sinal de realimentação de corrente do enrolamento de um motor em uma rede CAN, e que 0011 é a velocidade do motor, medida por um Tacômetro. Como 0010 tem o identificador binário mais baixo, as mensagens relativas aos valores atuais sempre têm uma prioridade mais alta no barramento do que aquelas relacionadas às leituras de velocidade.

Tipos de Mensagem

Os quatro tipos diferentes de mensagens que podem ser transmitidos em um barramento CAN são os frames de dados, o remoto, o de erro e o de sobrecarga. Nesse artigo vamos nos atentar ao frame de dados. Em outra oportunidade posso estender o assunto, mas, no contexto esse é o tipo que veremos.

O frame de dados é o tipo de mensagem mais comum e compreendem os campos de Arbitragem, o de Dados, o CRC e o de Reconhecimento. Na CAN padrão o campo de arbitragem contém um identificador de 11 bits e o bit RTR, que é dominante para os frames de dados. Na CAN estendida, ele contém o identificador de 29 bits e o bit RTR. Depois vem o campo de dados que contém de zero a oito bytes de dados e o campo CRC, que contém a soma de verificação de 16 bits usada para detecção de erros. Último é o campo de reconhecimento.

Verificação de Erros

A robustez da CAN pode ser atribuída em parte aos seus abundantes procedimentos de verificação de erros. O protocolo CAN incorpora métodos de verificação de erros. Se uma mensagem falha em qualquer um desses métodos de detecção de erros, ela não é aceita e um frame de erros é gerado a partir do nó de recebimento. Isso força o nó transmissor a re-enviar a mensagem até que ela seja recebida corretamente. No entanto, se um nó defeituoso, desliga o barramento, repetindo continuamente um erro, sua capacidade de transmissão é removida pelo controlador depois que um limite de erro é atingido.

Uma das maiores vantagens do protocolo CAN é a sua capacidade de se adaptar às condições de falha temporárias e/ou permanentes. Podemos classificar as falhas de uma rede CAN em três categorias ou níveis: Nível de Bit, Nível de Mensagem e Nível Físico. Em outra oportunidade posso estender o assunto, mas, para o artigo não ficar muito estenso vou parar por aui com os conceitos fundamentais de uma CAN.

Sobre a Aplicação

Depois de apresentados os conceitos de uma CAN, agora vamos focar na aplicação, veja novamente figura 1. Como o Arduino UNO não possui um periférico Controlador CAN, então, usamos um Controlador distinto, o MCP2515. Esse Controlador da Microchip é um dispositivo independente que controla uma comunicação CAN, implementado com a especificação CAN, versão 2.0B. Ele é capaz de transmitir e receber mensagens padrões e estendidas. O MCP2515 faz interface com os MCUs por meio de uma comunicação SPI padrão.
Também a saída do MCP2515 não é apropriada para um barramento CAN, portanto, um Transceiver é integrado ao projeto, nesse caso TJA1050. Veja os detalhes nas figuras a seguir. O jumper para o resistor de terminação (120 Ohms) foi fechado em ambas placas.

Figura 7 - Placa Controladora CAN + Transceiver. Extraída de um artigo técnico no site http://henrysbench.capnfatz.com

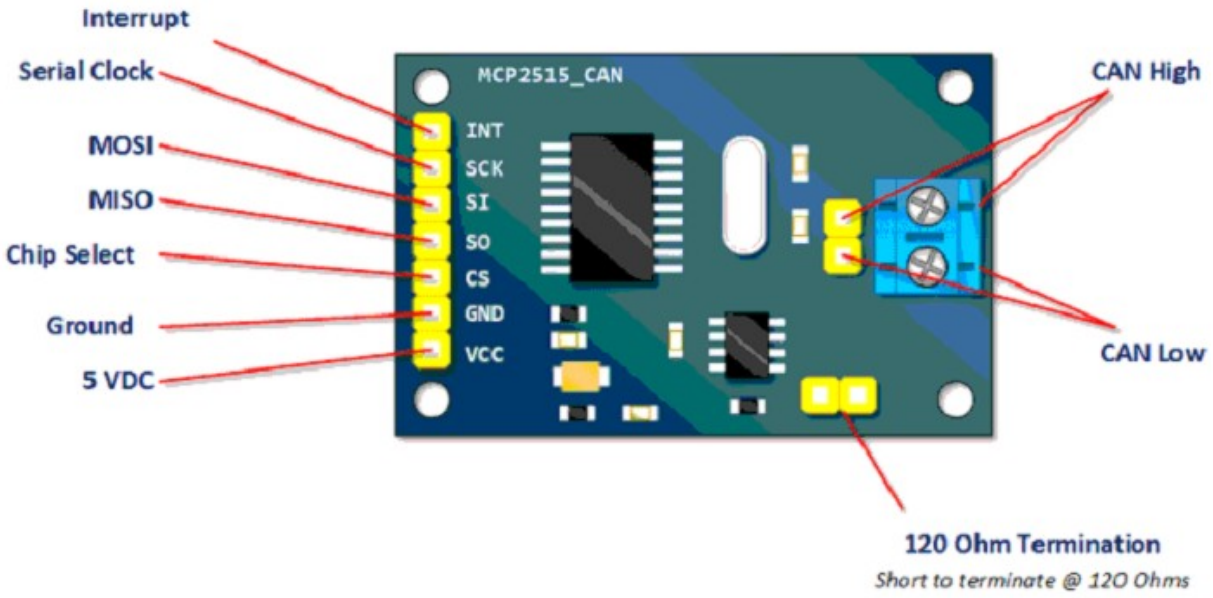
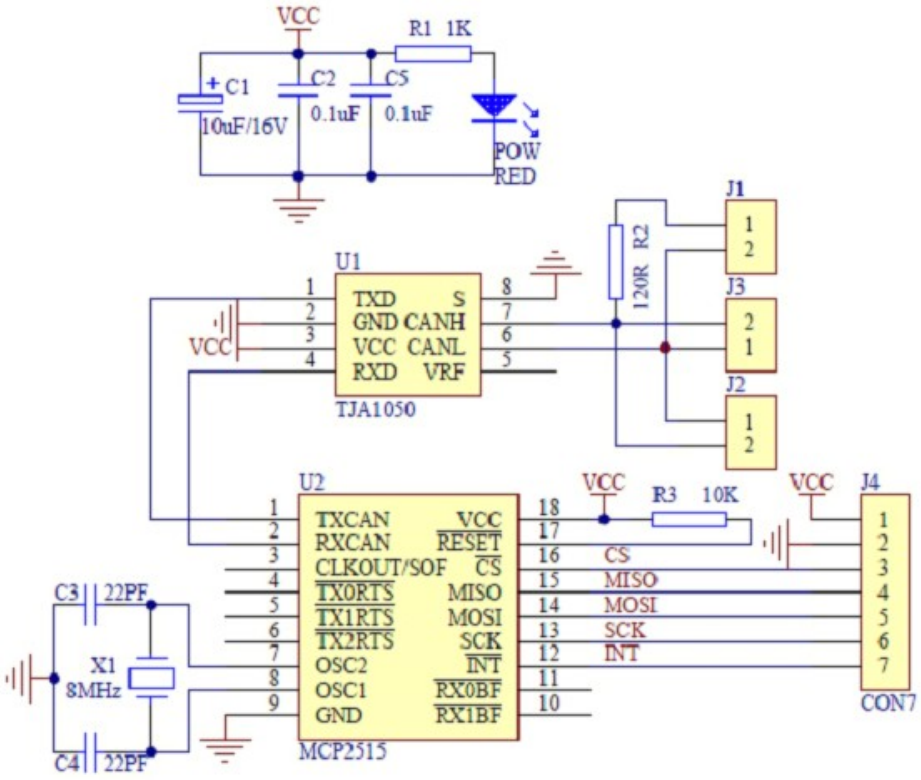


Figura 8 - Esquemático do Controlador CAN + Transceiver. Extraída de um anúncio no site https://produto.mercadolivre.com.br



Recurso Complementar

Para essa aplicação é preciso obter a biblioteca do MCP2515 e incluí-la na IDE do Arduino. Eu baixei essa biblioteca zipada no GitHub, link https://github.com/Seeed-Studio/CAN_BUS_Shield.

Aplicação Controlador #1

Segue o programa implementado na IDE do Arduino e carregado no Arduino #1:

```
//Para usar a bibliotecade comunicação SPI
#include <SPI.h>
//Para usar a biblioteca de comunicação CAN
#include <mcp2515.h>
//Pino de entrada que será o CS (Chip Select) da comunicação SPI entre Arduino e o Controlador CAN
MCP2515 mcp2515(10);
//Declara uma estrutura de dados com base numa estrutura predefinida na biblioteca do Controlador CAN
```

```
struct can_frame canMsg;
//Declara e inicializa o dado que esse controlador CAN transmitirá
int msgData0 = 0;

void setup()
{
  //Inicia a comunicação Serial
  Serial.begin(9600);
  //Inicia a comunicação SPI
  SPI.begin();
  //Reset do Controlador CAN atraves da SPI do Arduino
  mcp2515.reset();
  //Configura a velocidade de comunicação CAN para 500KBPS com um Clock de
8MHz. Clock do cristal do Controlador MCP2515
  mcp2515.setBitrate(CAN_500KBPS,MCP_8MHZ);
  //Modos de Operação do Controlador CAN: 1. Configuration mode 2. Normal mode
3. Sleep mode 4. Listen-Only mode 5. Loopback mode
  //Configura o modo normal
  mcp2515.setNormalMode();
}

void loop()
{
  // Se houver dado disponivel na Serial para ser lido
  while (Serial.available() > 0)
  {
    msgData0 = Serial.parseInt();
    Serial.println(msgData0);
    // Aguarda um caracter de controle de nova linha (newline)
    if (Serial.read() == '\n');
    canMsg.can_id  = 0x036;     //CAN id = 0x036
    canMsg.can_dlc = 1;         //CAN data length = 1 (pode ser no máximo 8 bytes)
    canMsg.data[0] = msgData0;  //CAN data0 = Dado lido na Serial
    //canMsg.data[1] = 0x00;      //CAN data1 = 0
    //canMsg.data[2] = 0x00;      //CAN data2 = 0
    //canMsg.data[3] = 0x00;      //CAN data3 = 0
    //canMsg.data[4] = 0x00;      //CAN data4 = 0
    //canMsg.data[5] = 0x00;      //CAN data5 = 0
    //canMsg.data[6] = 0x00;      //CAN data6 = 0
```

```
    //canMsg.data[7] = 0x00;      //CAN data7 = 0
    //Enviar a Mensagem CAN - Transceiver colocará essa mensagem no barramento
CAN
    mcp2515.sendMessage(&canMsg);
  }

  // Lê o dado de uma mensagem específica
  if (mcp2515.readMessage(&canMsg) == MCP2515::ERROR_OK)
  {
    if (canMsg.can_id == 0x035)
    {
      int x = canMsg.data[0];
      Serial.println(x);
    }
  }
  delay(1000);
}
```

Aplicação Controlador #2

Segue o programa implementado na IDE do Arduino e carregado no Arduino #2:

```
//Para usar a bibliotecade comunicação SPI
#include <SPI.h>
//Para usar a biblioteca de comunicação CAN
#include <mcp2515.h>
//Pino de entrada que será o CS (Chip Select) da comunicação SPI entre Arduino e
o Controlador CAN
MCP2515 mcp2515(10);
//Declara uma estrutura de dados com base numa estrutura predefinida na
biblioteca do Controlador CAN
struct can_frame canMsg;
//Declara e inicializa o dado que esse controlador CAN transmitirá
int msgData0 = 0;

void setup()
{
  //Inicia a comunicação Serial
  Serial.begin(9600);
  //Inicia a comunicação SPI
```

```
  SPI.begin();
  //Reset do Controlador CAN atraves da SPI do Arduino
  mcp2515.reset();
  //Configura a velocidade de comunicação CAN para 500KBPS com um Clock de
8MHz. Clock do cristal do Controlador MCP2515
  mcp2515.setBitrate(CAN_500KBPS,MCP_8MHZ);
  //Modos de Operação do Controlador CAN: 1. Configuration mode 2. Normal mode
3. Sleep mode 4. Listen-Only mode 5. Loopback mode
  //Configura o modo normal
  mcp2515.setNormalMode();
}

void loop()
{
  // Se houver dado disponivel na Serial para ser lido
  while (Serial.available() > 0)
  {
    msgData0 = Serial.parseInt();
    Serial.println(msgData0);
    // Aguarda um caracter de controle de nova linha (newline)
    if (Serial.read() == '\n');
    canMsg.can_id  = 0x035;     //CAN id = 0x035
    canMsg.can_dlc = 1;         //CAN data length = 1 (pode ser no máximo 8 bytes)
    canMsg.data[0] = msgData0;  //CAN data0 = Dado lido na Serial
    //canMsg.data[1] = 0x00;      //CAN data1 = 0
    //canMsg.data[2] = 0x00;      //CAN data2 = 0
    //canMsg.data[3] = 0x00;      //CAN data3 = 0
    //canMsg.data[4] = 0x00;      //CAN data4 = 0
    //canMsg.data[5] = 0x00;      //CAN data5 = 0
    //canMsg.data[6] = 0x00;      //CAN data6 = 0
    //canMsg.data[7] = 0x00;      //CAN data7 = 0
    //Enviar a Mensagem CAN - Transceiver colocará essa mensagem no barramento
CAN
    mcp2515.sendMessage(&canMsg);
  }

  // Lê o dado de uma mensagem específica
  if (mcp2515.readMessage(&canMsg) == MCP2515::ERROR_OK)
  {
```

```
    if (canMsg.can_id == 0x036)
    {
      int x = canMsg.data[0];
      Serial.println(x);
    }
  }
  delay(1000);
}
```

Como Funciona Essa Aplicação

Conforme a figura 1, temos uma CAN com dois Arduinos trocando mensagens. Para simplificar, as mensagens transmitem apenas um byte de dado, mas, a aplicação deixa claro que para transmitir até 8 bytes é muito simples (parte comentada). Um Arduino envia mensagem com o identificador 0x35, e somente envia um dado numérico de 0 à 255, que é entrado via terminal de comunicação serial, dentro da IDE do Arduino, então, a mensagem somente é colocada no barramento CAN quando o usuário, no terminal da IDE, digitar o dado e pressionar a tecla [ENTER]. Esse Arduino somente recebe a mensagem com identificador 0x35, que vem do outro Arduino.

O outro Arduino envia mensagem com o identificador 0x36, e somente envia um dado numérico de 0 à 255, que é entrado via terminal de comunicação serial, dentro da IDE do Arduino, então, a mensagem somente é colocada no barramento CAN quando o usuário, no terminal da IDE, digitar o dado e pressionar a tecla [ENTER]. Esse Arduino somente recebe a mensagem com identificador 0x36, que vem do outro Arduino.

Pensei em fazer dessa forma para que a rede fique livre e somente haverá troca de mensagem quando houver necessidade, portanto, se houvesse mais dispositivos pendurado na CAN esses dois Arduinos trocam mensagens sobre demanda. Como foi visto no conceito básico, obviamente todos os dispositivos na rede recebem as mensagens, mas, usando filtro o dado somente é consumido quando interessa. Essa é a beleza que está por trás de tudo, além da robustez e arbitragem. Com essa aplicação é possível experimentar os fundamentos aqui abordados. Espero que esse artigo seja útil a alguém, porque quando eu comecei a estudar a CAN ví muito coisa solta e não foi fácil chegar a esse ponto. Claro que tem muito mais a ser visto, porém, para atender o propósito acredito que seja um pontapé inicial.

Referências

1) Documento da Microchip 'MCP2515 Stand-Alone CAN Controller with SPI Interface' disponibilizado no site da Microchip.
2) Documento da Texas Instruments 'Introduction to the Controller Area Network (CAN)' Application Report SLOA101B–August 2002–Revised May 2016 by Steve Corrigan.
3) Documento da Philips 'TJA1050 High speed CAN transceiver' Datasheet Product specification Supersedes data of 2002 May 16.
4) Site https://www.electronicshub.org/arduino-mcp2515-can-bus-tutorial.
5) Site https://github.com/Seeed-Studio/CAN_BUS_Shield.