Alejandro Ossio Díaz, A01209122
Gerardo Daniel Naranjo Gallegos, A01209499
Roberto Figueroa Saavedra, A01209689
28/11/2018

Proyecto CANBUS

Contenido

1. Introducción

En la actualidad, los vehículos cuentan con una gran cantidad de sensores y actuadores; entre mayor la gama del vehículo, mayor el número de sensores con los que cuenta. Para lograr la comunicación entre cada sensor, actuador y la computadora del vehículo, se utilizan cables. Dichos cables, pueden corresponder a distintos protocolos de comunicación, pero actualmente el más implementado y que presenta mayores beneficios es el protocolo de comunicación llamado CANBUS.

Para poner en práctica los conocimientos teóricos sobre CANBUS vistos en clase, se realiza la siguiente práctica que consta de efectuar una conexión, tanto serial como CAN, con el objetivo de simular la comunicación que se efectúa en un vehículo actual en el sistema de iluminación exterior.

2. Material utilizado

El material utilizado para realizar esta práctica se enlista a continuación:

- Cuatro Arduino Uno.
- Una laptop con IDE Arduino.
- Cuatro cables (USB-Arduino).
- Cable CANBUS.
- Un *shield* joystick.
- Dos *shield* CANBUS.
- Un *shield power* driver.
- Una Fuente de alimentación de 12V.
- Cuatro intermitentes (dos bobinas y dos LED).
- Un faro principal.
- Un LED.
- Seis cables dupont macho-macho.
- Un *push-button*.
- Un protoboard.

3. Descripción de la práctica

A lo largo de esta práctica, se explicará cómo controlar cuatro intermitentes (dos focos de bobina y dos focos LED) y un faro principal, con función de luz alta y luz baja, en simulación del sistema de iluminación externa de un vehículo (excluyendo luces de freno y luces de posición). Actualmente, los vehículos cuentan con el protocolo de comunicación CANBUS para estos sistemas. Para lograrlo, se deberán conectar conforme a los esquemáticos, encontrados al final del documento en la sección de anexos.

La práctica consiste en cuatro etapas, una para cada Arduino utilizado. La primera referente al primer Arduino, portador del *shield* de *SparkFun Joystick*, que servirá como la interfaz que permita controlar qué luces encender. Éste Arduino, se comunicará con el segundo Arduino (o segunda etapa) mediante comunicación serial.

La segunda y tercera etapa, que contemplan al segundo y tercer Arduino, llevarán el *shield* de Arduino CANBUS. A partir de ahora, a la segunda etapa se le llamará al Arduino que recibe mediante serial del primer Arduino y tendrá la función de transmitir mediante el CAN qué luces encender/apagar. Se le llamará tercera etapa al Arduino con la función de recibir el mensaje del segundo Arduino mediante CAN, para transmitir mediante serial al cuarto Arduino (o cuarta etapa).

Posteriormente a recibir mediante serial del tercer Arduino, el cuarto Arduino tendrá la función de encender/apagar las luces de la forma indicada desde el primer Arduino.

Como podemos observar, durante la segunda y tercera etapa, se implementará el protocolo de CANBUS para ponerlo en práctica. La primera y cuarta etapa funcionan a través de comunicación serial. Para información más detallada del protocolo CAN y las funcionalidades de cada etapa del proyecto, consultar las secciones posteriores.

4. Protocolo CANBUS:

En esta práctica se implementará comunicación utilizando Bus CAN. CAN proviene del inglés *Controller Area Network*, es un protocolo de comunicaciones.

Una característica interesante de este protocolo es que cada esclavo puede pasar a ser maestro. Además, presenta diversas ventajas, como una alta inmunidad al ruido (perturbaciones/interferencias) y el economizar, ya que el cableado necesario es considerablemente menor, por ser una red multiplexada. Además, se presentan ventajas como la prioridad en mensajes, tiempos de latencia y una gran distinción de errores.

Físicamente, el CANBJS es un par de cable trenzado, como se puede observar en la siguiente imagen:

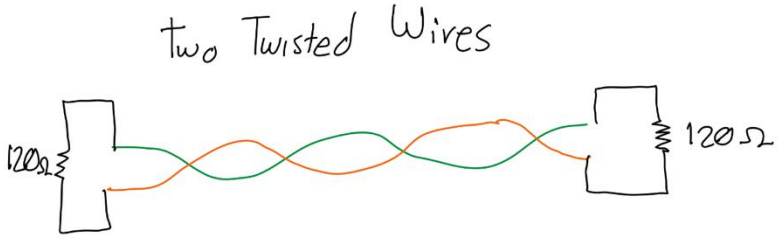

Figura 1. Tipo de cables del protocolo CAN.

Continuando con el apartado físico, podemos observar cómo se identifican los estados lógicos en la siguiente imagen.

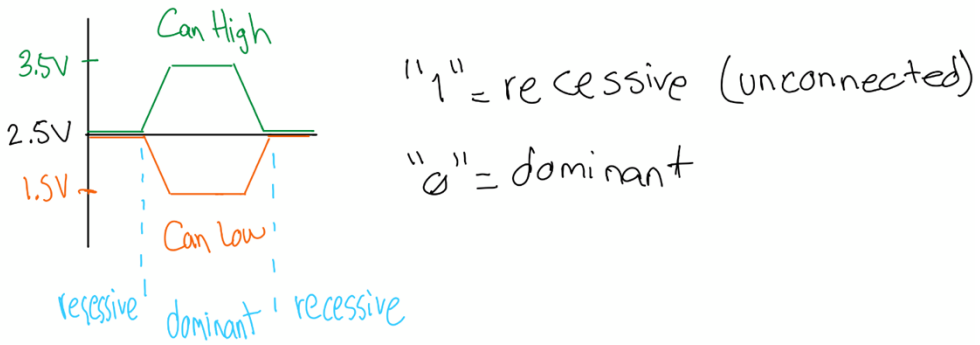

Figura 2. Estados lógicos del protocolo CAN.

El protocolo CAN utiliza la siguiente estructura para su paquete de datos:

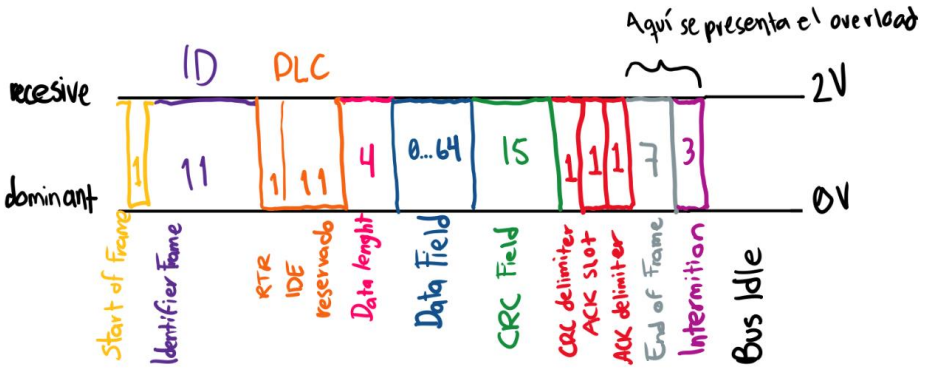

Figura 3. Paquete de datos de CAN.

5. SparkFun joystick shield for Arduino

Este Arduino, corresponde a la primera etapa del proyecto. Consiste de un Arduino, el cual cuenta con un *shield* montado, se puede apreciar en el esquemático de la sección 13.1 y cuenta con el código presentado en la sección 13.2.

El *shield* implementado en este Arduino, presenta un *joystick* y cuatro botones (*push button*), que serán utilizados para realizar las distintas funciones, como se explica a continuación.

- Botón del *joystick*: activa/desactiva las intermitentes. Conectado al pin D2 del Arduino.
- Botón superior: activa/desactiva las luces altas. Conectado al pin D4 del Arduino.
- Botón inferior: activa/desactiva las luces bajas. Conectado al pin D5 del Arduino.
- Botón izquierdo: activa/desactiva las intermitentes del lado izquierdo. Conectado al pin D6 del Arduino.
- Botón derecho: activa/desactiva las intermitentes del lado derecho. Conectado al pin D3 del Arduino.

Adicionalmente, se implementa en un circuito externo un botón extra. El botón solamente va conectado a alimentación y envía la señal al pin D7 del Arduino. Este botón adicional activa la luz de cortesía; dicha función es un extra en la práctica. La luz de cortesía, o también conocida como función de *coming home* o *leaving home*, consiste en encender las luces del vehículo (cuando está apagado), con el propósito de iluminar por unos segundos el camino desde o hacia el vehículo.

La ocupación de este primer Arduino será reconocer qué botón ha sido oprimido, para mandar al Arduino de la etapa dos, mediante comunicación serial, las instrucciones necesarias. Previo a esto, se comprueban ciertas validaciones:

- Al activar las luces altas, se deben de desactivar las luces bajas. Esto debido a que la fuente utilizada no soportaría alimentar a ambas.
- Al activar una direccional, debe encender; si previamente estaba encendida la direccional opuesta, esta deberá apagarse.
- Al encender las intermitentes, tendrá prioridad sobre la función de direccionales. Si se encontraba encendida alguna direccional, esta deberá de volver a encenderse una vez apagadas las intermitentes. Además, si se apaga o se realiza un cambio de direccional mientras las intermitentes están

encendidas, al apagarse las intermitentes se debe respetar este cambio o apagado.

- La luz de cortesía solamente debe encender el faro de la luz baja durante 10 segundos. Esta luz solamente encenderá si todo lo demás está previamente apagado (emulando que el carro estuviese apagado).

Todas las validaciones se llevan a cabo en el código del Arduino, mediante la implementación de funciones como *if*, por ejemplo.

Como validación extra, también se debe de considerar el llamado rebote mecánico del botón. En este caso, se resolvió mediante software, utilizando banderas para saber cuándo es oprimido (y soltado) el botón.

En cuanto a la comunicación serial, primeramente, se genera una cadena de 8 caracteres; en donde se respeta el siguiente orden y donde un "0" representa apagado y un "1" representa encendido:

0) "#": simboliza el inicio de la cadena y siempre es el mismo carácter.
1) "0" o "1": representa a las intermitentes.
2) "0" o "1": simboliza a la direccional derecha.
3) "0" o "1": figura a la direccional izquierda.
4) "0" o "1": utilizado para las luces altas.
5) "0" o "1": requerido para las luces bajas.
6) "0" o "1": implementado para la luz de cortesía.
7) "&": simboliza el fin de la cadena y siempre es el mismo carácter.

Posteriormente, con la información de qué luces encender, se envía la cadena de caracteres al Arduino de la segunda etapa mediante comunicación serial.

Para lograr la comunicación serial entre ambos Arduinos, es necesario iniciarla y ajustar el *baud rate* (como se puede observar en el código) y conectar los puertos TX y RX (con los cables dupont). Los puertos de transmisión y recepción se conectan de forma TX con RX y viceversa, es decir, un puerto transmisor se conecta al puerto receptor del otro Arduino. Finalmente, conectar la tierra entre ambos Arduinos.

Cabe mencionar que las conexiones físicas de TX y RX se deben de desconectar cada que se desea reprogramar uno de los dos Arduinos. De no hacerlo, existe la posibilidad de dañarlos irremediablemente.

6. *Arduino CANBUS shield* transmisor

Este Arduino cumple la función de recibir, mediante comunicación serial, la cadena de caracteres del Arduino de la etapa 1, posteriormente, mandarla mediante CANBUS al Arduino de la etapa 3.

Para lograr la comunicación serial, basta con conectar los puertos TX y RX del Arduino 1 y del Arduino 2.

En cuanto a la comunicación CANBUS, es requerido el *shield*. Para una mayor información, se muestra el esquemático en la sección 13.3.

El código del Arduino se puede encontrar en la sección 13.4. Básicamente, el Arduino debe de recibir la cadena de caracteres y reenviarla mediante CANBUS al Arduino tres, de la tercera etapa. Para esto se sigue un tipo de mensaje predefinido, en el cual lo primero que se envía el tipo de formato en que se va a enviar, en este caso nosotros estamos utilizando un 0x631 el cual corresponde a un formato hexadecimal, después lleva un *header* en el cual se especifica el número de bytes que se van a enviar, esto es equivalente a la longitud del mensaje, posteriormente se envía cada uno de los datos en el formato establecido y se finaliza el mensaje.

7. *Arduino CANBUS shield* receptor

Este Arduino recibe el mensaje mandado por el anterior mediante comunicación CAN, el código se puede encontrar en la sección 13.5, el cual se encarga de desempaquetar el mensaje CAN, leyendo el tipo de formato, después el número de bytes que se van a leer y hace un corrimiento para guardar cada uno en un arreglo, que posteriormente se convertirá para enviarlo mediante serial al próximo Arduino.

8. *SparkFun power driver shield for Arduino*

Este último Arduino, cuenta con el *shield* descrito en la Figura 6 y con el código de la sección 13.7. Su función es recibir la cadena de caracteres generada desde el primer Arduino, mediante comunicación serial, para poder identificar carácter por carácter y encender las luces correspondientes. Cabe mencionar que las validaciones se realizan desde el primer Arduino.

Esta etapa cuenta con una fuente de alimentación. En este caso particular, es una fuente de PC. Por lo tanto, se debe verificar que la fuente esté encendida para su correcto funcionamiento; es la alimentación de los faros e intermitentes.

Adicionalmente, este *shield* cuenta con un *switch* que se deberá encender.

La lógica del código es la inversa que la del Arduino en la primera etapa. Es decir, descompone la cadena de caracteres formada. Con base en cada carácter, se identifica cuáles luces deberán de ser encendidas. De esta forma, se manda la señal a través de los pines:

- D3: direccional izquierda trasera.
- D5: luz del faro baja.
- D6: direccional derecha delantera.
- D8: luz adicional agregada para identificar fallas.
- D9: direccional izquierda delantera.
- D10: direccional trasera derecha.
- D11: luz del faro alta.

Para mayor información, se puede consultar el esquemático de la sección 13.6.

9. Pasos para llevar a cabo

El método de elaboración de esta práctica consiste, como se ha dicho previamente, en cuatro etapas.

La primera etapa por llevar a cabo es programar la programación de todos los Arduinos con su respectivo código, ubicados en la sección de Anexos.

Posteriormente, a cada Arduino se le debe montar su *shield* necesario.

Consecutivamente, se deben de conectar los puertos necesarios, por ejemplo, en los pines de TX, RX y GND (tierra) de los Arduinos que tendrán la conexión serial. Es decir, el primero con el segundo y el tercero con el cuarto.

Además, conectar los cables de comunicación CANBUS entre el segundo y tercer Arduino.

Finalmente, conectar los cables de alimentación de los cuatro Arduinos y encender el HUB de alimentación.

Particularmente para el cuarto Arduino, se cuenta con la fuente de voltaje que alimentará a los faros. Dicha fuente, deberá de ser encendida. Después, se deberá colocar en la posición de encendido el *switch* con el que cuenta el case de dicho Arduino.

10. Resultados obtenidos

Los resultados del proyecto funcional es el control deseado del faro, en altas y bajas, y los cuatro intermitentes, con la función de direccionales e intermitentes. Contemplando en todos los casos sus respectivas validaciones e, incluso, con la implementación de extras.

11. Principales desafíos

Como principales retos a lo largo de la práctica realizada, se puede mencionar a las diversas validaciones a considerar, desde el primer Arduino. Más específicamente, las validaciones referentes a las intermitentes y direccionales.

Adicionalmente, los extras añadidos a la práctica, los cuales son:

- Lograr que las intermitentes y direccionales enciendan al mismo tiempo: Esto, debido a que las intermitentes traseras eran utilizadas en forma de LED y las delanteras con faros convencionales, por lo tanto, las traseras encendían primero, debido a que los faros LED no requieren calentar filamento, a diferencia de los convencionales. La solución se logró mediante la implementación de la función *delay* en el código.
- La implementación de una luz de cortesía: representó desafío lograr que la luz se encienda en 10 segundos, debido a factores internos del Arduino. Se solucionó con la función *delay* en el código.
- Finalmente, se incluyó una luz que indicaba algún error presentado: tratando de simular un testigo de *check engine* del clúster de instrumentos. El reto fue diseñar el circuito externo, aunque sencillo, que contemplara el *push button* adicional.

12. Referencias

García, A. (24 de junio de 2015). *Diseño de una red CAN bus con Arduino .* Obtenido de E.T.S. de Ingeniería Industrial, Informática y de Telecomunicación | Universidad Pública de Navarro: http://academica-e.unavarra.es/xmlui/bitstream/handle/2454/19115/TFG%20Diseño%20de%20una%20Red%20Can%20bus%20-%20Alejandro%20García%20Osés.pdf?sequence=1

GitHub. (18 de enero de 2017). *SparkFun_CAN-Bus_Arduino_Library .* Obtenido de GitHub: https://github.com/sparkfun/SparkFun_CAN-Bus_Arduino_Library

13. Anexos

13.1. Esquemático SparkFun *joystick shield*:

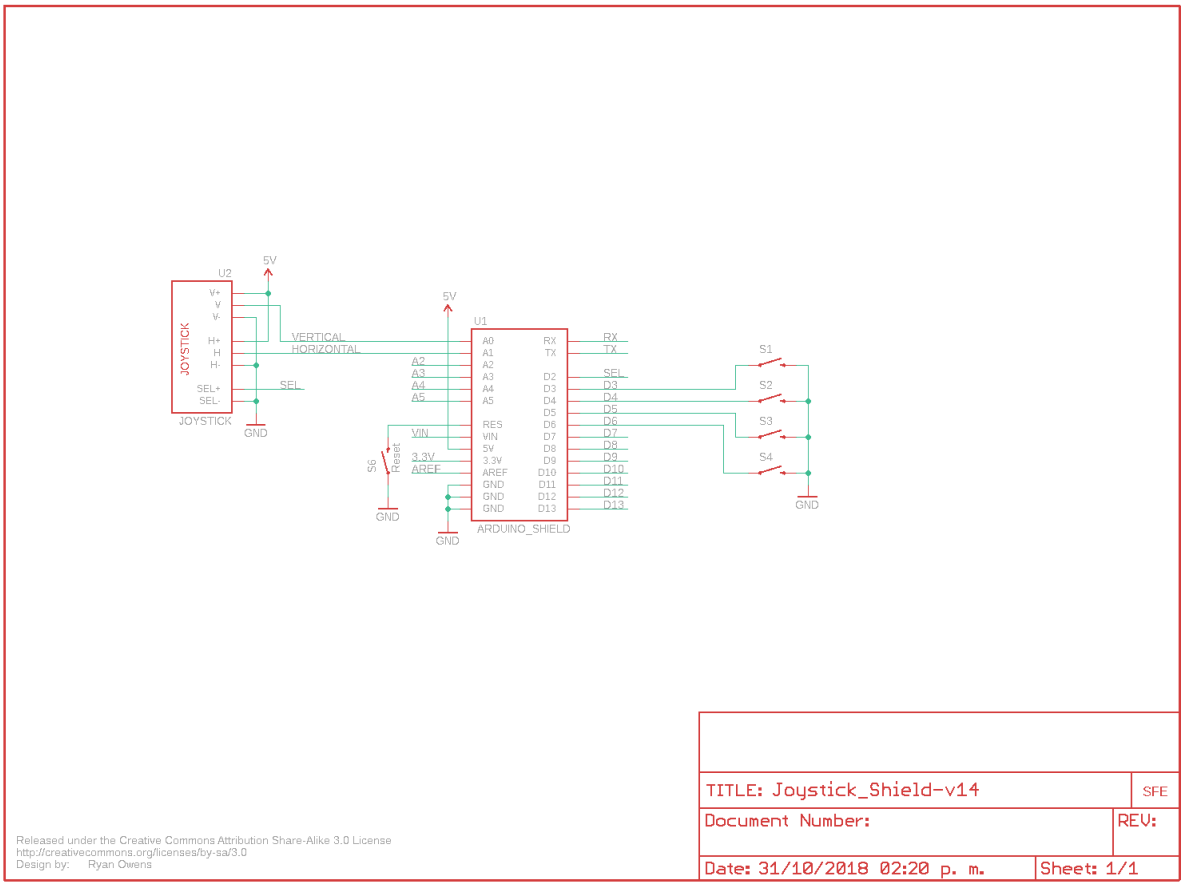

Figura 4. Esquemático *Joystick Shield*.

13.2. Código para Arduino con SparkFun *joystick shield*:

```
//Librerías utilizadas:
int D2 = 2;    //HAZARD
int D3 = 3;    //TURNR
int D6 = 6;    //TURNL
int D4 = 4;    //HIGHB
int D5 = 5;    //LOWB
int D7 = 7;    //COURTESY


  //State flag
  int HAZARD = 0, TURNR = 0, TURNL = 0, HIGHB = 0, LOWB = 0, COURTESY =
0;
  //Previous state for turns
  int PrevTurnR = 0, PrevTurnL = 0, PrevHazard = 0;
  //Mechanical bounce flag
  int D4R = 0, D5R = 0, D2R = 0, D3R = 0, D6R = 0, D7R = 0;
  //String to send
  char CAN[8] = {'0','0','0','0','0','0','0','0'};
  //Change of state
  bool match = true;
  //Counter for buttons
  int Count2 = 0, Count3 = 0, Count4 = 0, Count5 = 0, Count6 = 0, Count7
= 0, TotalCount = 0, Ciclado = 0;
  //Variable for changes in turn while in hazard
  int HTurn = 0;

void setup() {
  int i;
 for(i = 2; i < 8; i++){
  pinMode(i,INPUT);
  digitalWrite(i,HIGH);
  }
  Serial.begin(9600);
}

//Method to convert int to char:
char ConvertIntToChar(int entero){
  switch(entero)
  {
    case 1:
      return '1';
      break;
    case 0:
      return '0';
      break;
    default:
      return '?';
      break;
  }
}

//Method to build the matrix to send
void MakeOutput()
{
```

```
  CAN[0] = '#';
  CAN[1] = ConvertIntToChar(HAZARD);
  CAN[2] = ConvertIntToChar(TURNR);
  CAN[3] = ConvertIntToChar(TURNL);
  CAN[4] = ConvertIntToChar(HIGHB);
  CAN[5] = ConvertIntToChar(LOWB);
  CAN[6] = ConvertIntToChar(COURTESY);
  CAN[7] = '&';
}

void loop() {
  if(digitalRead(D2) == LOW){
    D2R = 1;
    Count2 = 1;
  }
   delay(30);
  if(digitalRead(D4) == LOW){
    D4R = 1;
    Count4 = 1;
  }
   delay(30);
  if(digitalRead(D5) == LOW){
    D5R = 1;
    Count5 = 1;
  }
   delay(30);
  if(digitalRead(D3) == LOW){
    D3R = 1;
    Count3 = 1;
  }
  delay(30);
  if(digitalRead(D6) == LOW){
    D6R = 1;
    Count6 = 1;
  }
  delay(30);


  if(HAZARD == 0 && TURNR == 0 && TURNL == 0 && HIGHB == 0 && LOWB == 0){
      if(digitalRead(D7) == LOW){
        D7R = 1;
        Count7 = 1;
      }
  delay(30);
    if(digitalRead(D7) == HIGH && D7R == 1){
      COURTESY = 1;
      match = false;
      D7R = 0;
      Count7 = 0;
      }
  }

if(COURTESY == 0){
  if(digitalRead(D4) == HIGH && D4R == 1){
    LOWB = 0;
    HIGHB = !HIGHB;
    D4R = 0;
```

```
  match = false;
    Count4 = 0;
}
if(digitalRead(D5) == HIGH && D5R == 1){
  HIGHB = 0;
  LOWB = !LOWB;
  D5R = 0;
  match = false;
    Count5 = 0;
}
if(digitalRead(D2) == HIGH && D2R == 1){
  TURNR = 0;
  TURNL = 0;
  HAZARD = !HAZARD;
  D2R = 0;
  PrevHazard = 1;
  match = false;
    Count2 = 0;
}

if(HAZARD == 0){

    if(PrevHazard == 1){
      TURNR = PrevTurnR;
      TURNL = PrevTurnL;
    }

    if(HTurn == 1){
      TURNR = PrevTurnR;
      TURNL = PrevTurnL;
    }

  if(digitalRead(D3) == HIGH && D3R == 1){
    TURNL = 0;
    TURNR = !TURNR;
    D3R = 0;
    match = false;
    Count3 = 0;
  }
  if(digitalRead(D6) == HIGH && D6R == 1){
    TURNR = 0;
    TURNL = !TURNL;
    D6R = 0;
    match = false;
    Count6 = 0;
  }

  PrevTurnR = TURNR;
  PrevTurnL = TURNL;
  PrevHazard = 0;
  HTurn = 0;
}
else if(HAZARD == 1){
 if(digitalRead(D3) == HIGH && D3R == 1){
    PrevTurnL = 0;
    PrevTurnR = !PrevTurnR;
    D3R = 0;
```

```
      match = false;
      Count3 = 0;
    }
    if(digitalRead(D6) == HIGH && D6R == 1){
      PrevTurnR = 0;
      PrevTurnL = !PrevTurnL;
      D6R = 0;
      match = false;
      Count6 = 0;
    }
    HTurn = 1;
  }


}

  TotalCount = Count2 + Count3 + Count4 + Count5 + Count6 + Count7;

  if(TotalCount >= 2){
      CAN[8] = "#000000&";
  }


  MakeOutput();
  if(match == false)
    if(Serial.availableForWrite())
      Serial.write(CAN,8);
  match = true;
  COURTESY = 0;
}
```

13.3. Esquemático Arduino CANBUS *shield*:

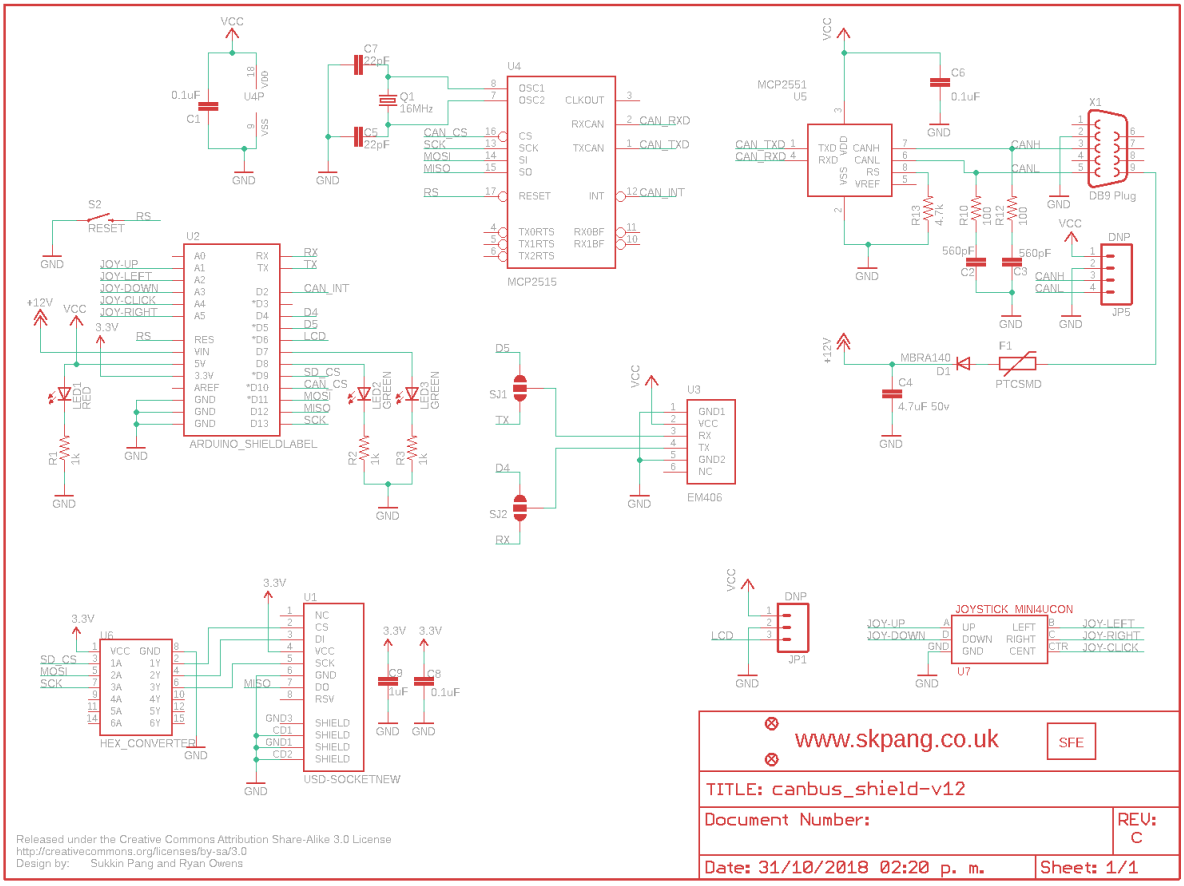

Figura 5. Esquemático CANBUS *Shield*.

13.4. Código para Arduino con SparkFun *CANBUS shield transmisor*:

```
#define BAUDRATE  9600

#include <Canbus.h>
#include <defaults.h>
#include <global.h>
#include <mcp2515.h>
#include <mcp2515_defs.h>

void setup() {
  Serial.begin(BAUDRATE);
  Canbus.init(CANSPEED_500);
}

//Variables;
char InputChar[8]    = {'x','x','x','x','x','x','x','x'};
bool Can             = false;

//Método de comunicación:
void Communication()
```

```
{
  if(Serial.available() > 0)
  {
    Serial.readBytes(InputChar, 8);
    Can = true;
  }
  else
    Can = false;
}

void Debug()
{
  Serial.print("Cadena = ");
  Serial.println(InputChar);
}

int ConvertionCharToHex(char toConvert)
{
  switch(toConvert)
  {
    case '0':
      return 0x30;
      break;
    case '1':
      return 0x31;
      break;
    case '2':
      return 0x32;
      break;
    case '#':
      return 0x23;
      break;
    case '&':
      return 0x26;
      break;
  }
}

void CAN()
{
  tCAN message;
  message.id = 0x631;         //formatted in HEX
  message.header.rtr = 0;
  message.header.length = 7;  //formatted in DEC
  message.data[0] = ConvertionCharToHex(InputChar[0]);
  message.data[1] = ConvertionCharToHex(InputChar[1]);
  message.data[2] = ConvertionCharToHex(InputChar[2]);
  message.data[3] = ConvertionCharToHex(InputChar[3]);
  message.data[4] = ConvertionCharToHex(InputChar[4]);
  message.data[5] = ConvertionCharToHex(InputChar[5]);
  message.data[6] = ConvertionCharToHex(InputChar[6]);
  mcp2515_bit_modify(CANCTRL, (1<<REQOP2)|(1<<REQOP1)|(1<<REQOP0), 0);
  mcp2515_send_message(&message);
}

void loop()
{
```

```
  Communication();
  if(Can){
    Debug();
    CAN();
  }
}
```

13.5. Código para Arduino con SparkFun *CANBus shield receptor*:

```
#define BAUDRATE  9600

#include <Canbus.h>
#include <defaults.h>
#include <global.h>
#include <mcp2515.h>
#include <mcp2515_defs.h>

char Output[8] = {'x','x','x','x','x','x','x','x'};

void setup() {
  Serial.begin(BAUDRATE);
  Canbus.init(CANSPEED_500);
}

char ConvertionHexToChar(int toConvert)
{
  switch(toConvert)
  {
    case 0x30:
      return '0';
      break;
    case 0x31:
      return '1';
      break;
    case 0x32:
      return '2';
    case 0x26:
      return '&';
      break;
    case 0x23:
      return '#';
      break;
  }
}

void ReadCAN()
{
  tCAN message;
  if (mcp2515_check_message())                                    //Check for
available message (recibing from CANBUS Joystick)
  {
    if (mcp2515_get_message(&message))                            //Read the
message
    {
      for(int i=0;i<message.header.length;i++)
      {
          Output[i] = ConvertionHexToChar(message.data[i]);   //Saving
the convertion from HEX to Char in Array "Output" the message
      }
    }
    Serial.write(Output,8);                                       //Write in
Serial the message in Char
  }
```

```
}

void loop()
{
  ReadCAN();
}
```

13.6. Esquemático SparkFun *power driver shield*:

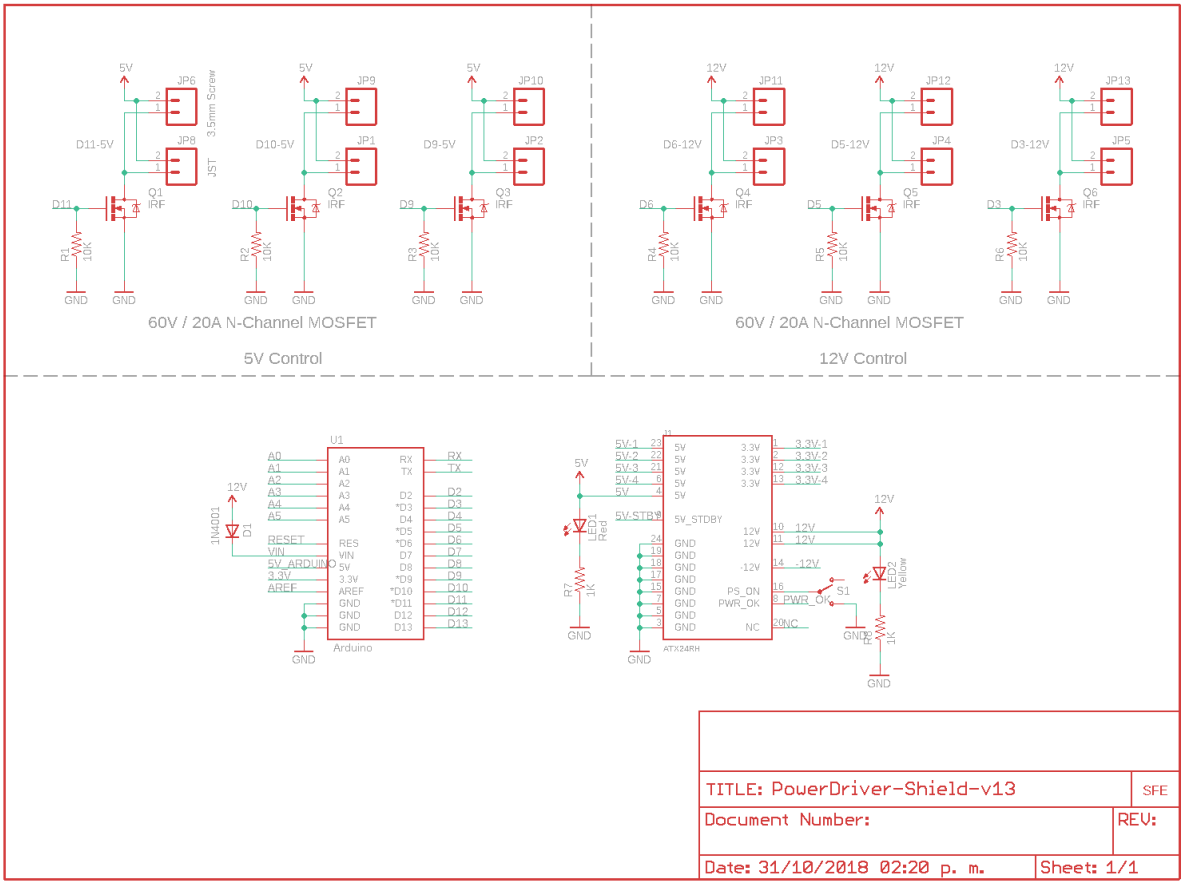

Figura 6. Esquemático *power driver shield*.

13.7. Código para Arduino con SparkFun *power driver shield*:

```
#define pin_FrontRight 6
#define pin_FrontLeft 9
#define pin_BackRight 10
#define pin_BackLeft 3
#define pin_HighBeam 11
#define pin_LowBeam 5
#define pin_Warning 8

#define BAUDRATE  9600

char InputChar[8] = {'x','x','x','x','x','x','x','x'};
bool Hazard = false;
bool TurnLeft = false;
bool TurnRight = false;
bool HighBeam = false;
bool LowBeam = false;

void setup() {
  Serial.begin(BAUDRATE);
```

```
  pinMode(pin_FrontRight, OUTPUT);
  pinMode(pin_FrontLeft, OUTPUT);
  pinMode(pin_BackRight, OUTPUT);
  pinMode(pin_BackLeft, OUTPUT);
  pinMode(pin_HighBeam, OUTPUT);
  pinMode(pin_LowBeam, OUTPUT);
  pinMode(pin_Warning, OUTPUT);
}

void Debug()
{
  Serial.print("Cadena = ");
  Serial.println(InputChar);
}

void Communication()
{
  if(Serial.available() > 0)
  {
    Serial.readBytes(InputChar, 8);
    Debug();
  }
}

void HAZARD()
{
  digitalWrite(pin_FrontRight, HIGH);
  delay(25);
  digitalWrite(pin_BackRight, HIGH);
  digitalWrite(pin_FrontLeft, HIGH);
  delay(25);
  digitalWrite(pin_BackLeft, HIGH);
  delay(500);
  digitalWrite(pin_FrontRight, LOW);
  delay(25);
  digitalWrite(pin_BackRight, LOW);
  digitalWrite(pin_FrontLeft, LOW);
  delay(25);
  digitalWrite(pin_BackLeft, LOW);
  delay(500);
}

void NOHAZARD(){
  digitalWrite(pin_FrontRight, LOW);
  digitalWrite(pin_BackRight, LOW);
  digitalWrite(pin_FrontLeft, LOW);
  digitalWrite(pin_BackLeft, LOW);
  }

void TURNLEFT()
{
  digitalWrite(pin_FrontLeft, HIGH);
  delay(25);
  digitalWrite(pin_BackLeft, HIGH);
  delay(500);
  digitalWrite(pin_FrontLeft, LOW);
  delay(25);
```

```
  digitalWrite(pin_BackLeft, LOW);
  delay(500);
}
void NOTURNLEFT(){
  digitalWrite(pin_FrontLeft, LOW);
  digitalWrite(pin_BackLeft, LOW);
  }

void TURNRIGHT()
{
  digitalWrite(pin_FrontRight, HIGH);
  delay(25);
  digitalWrite(pin_BackRight, HIGH);
  delay(500);
  digitalWrite(pin_FrontRight, LOW);
  delay(25);
  digitalWrite(pin_BackRight, LOW);
  delay(500);
}
void NOTURNRIGHT(){
  digitalWrite(pin_FrontRight, LOW);
  digitalWrite(pin_BackRight, LOW);

  }

void HIGHBEAM()
{
  digitalWrite(pin_HighBeam, HIGH);
  digitalWrite(pin_LowBeam, LOW);
}

void NOHIGHBEAM(){
  digitalWrite(pin_HighBeam, LOW);
  }

void LOWBEAM()
{
  digitalWrite(pin_LowBeam, HIGH);
  digitalWrite(pin_HighBeam, LOW);
}

void NOLOWBEAM(){
  digitalWrite(pin_LowBeam, LOW);
  }

void Validations()
  {
    if((InputChar[1] == '0' || InputChar[1] == '1') && InputChar[2] ==
'0' && InputChar[3] == '0' && InputChar[4] == '0' && InputChar[5] == '0'
){
      digitalWrite(pin_FrontRight, LOW);
      digitalWrite(pin_BackRight, LOW);
      digitalWrite(pin_FrontLeft, LOW);
      digitalWrite(pin_BackLeft, LOW);
      digitalWrite(pin_LowBeam, LOW);
      digitalWrite(pin_HighBeam, LOW);
    }
```

```
    if(InputChar[1] == '1'){
        HAZARD();
      }
    else if (InputChar[1] == '0'){
      NOHAZARD();
      }
    if(InputChar[2] == '1'){
        TURNRIGHT();
      }
    else if (InputChar[2] == '0'){
      NOTURNRIGHT();
      }
    if(InputChar[3] == '1'){
        TURNLEFT();
      }
    else if (InputChar[3] == '0'){
      NOTURNLEFT();
      }
    if(InputChar[4] == '1'){
        HIGHBEAM();
      }
    else if (InputChar[4] == '0'){
      NOHIGHBEAM();
      }
    if(InputChar[5] == '1'){
        LOWBEAM();
      }
    else if (InputChar[5] == '0'){
      NOLOWBEAM();
      }
  }

void loop() {
  Communication();
  if(InputChar[0] == '#' && InputChar[7] == 'x'){
    Validations();
    digitalWrite(pin_Warning,LOW);
  }
  else
    digitalWrite(pin_Warning,HIGH);
}
```