**AEL ZG513 – Automotive Communications**

**LAB MANUAL**



**Introduction**

This document explains different connectivity mechanisms in automotive systems that allow modules to exchange data in a bus. In order to illustrate data exchange between modules two nodes are connected using the following methods of communication sequentially. The circuits for connection, software coding using Arduino platform are illustrated and exercises provided where the student can alter the sequence of existing code to demonstrate and test the understanding of connectivity.

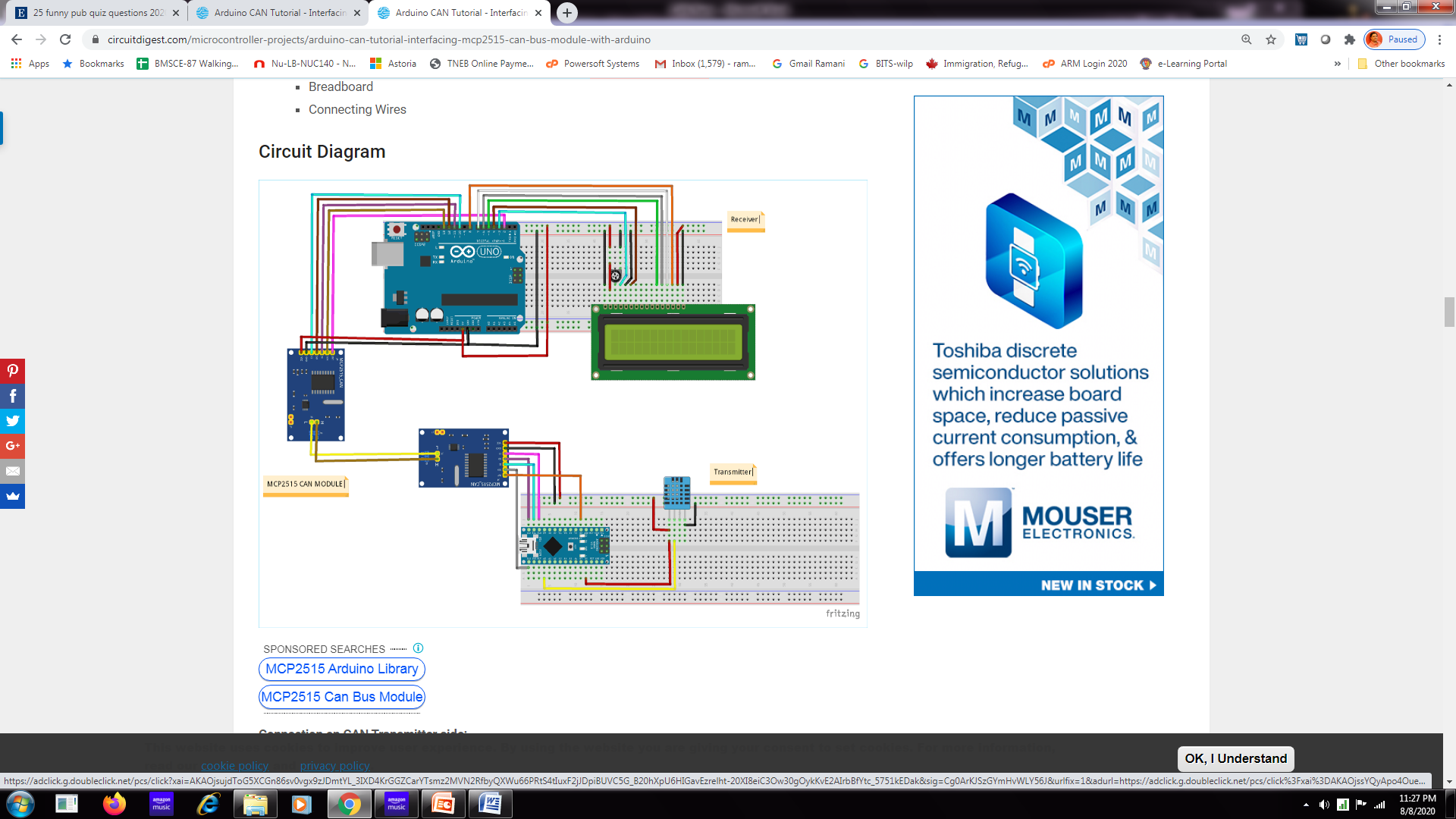
1. Ethernet
2. CAN
3. Bluetooth

**Ethernet Connectivity**

|  |  |  |
| --- | --- | --- |
| The Arduino board communicates with the Ethernet module using Serial Peripheral Interface (SPI bus). This means that the Arduino controller passes all relevant data to the Ethernet module such as setting up the module, configuring the module, passing data to be transmitted, reading data from the module using essentially the following four pins:  Arduino Pins  13- SCK – Serial Clock  12 - MISO- Master Input Slave Output  11- MOSI – Master Output Slave Input  10 SS – Slave Select  The Arduino Ethernet library is designed to work with the Arduino Ethernet Shield, Arduino Ethernet Shield 2, Leonardo Ethernet, and any other W5100/W5200/W5500-based devices. The library allows an Arduino board to connect to the Internet. The board can serve as either a server accepting incoming connections or a client making outgoing ones. The library supports up to eight concurrent connections (incoming, outgoing, or a combination).  The Arduino Uno board communicates with the shield using the SPI bus at pins - 11, 12, and 13 on the Uno and pins 50, 51, and 52 on the Arduino Mega. On both boards, pin 10 is used as SS. On the Arduino Mega, the hardware SS pin, 53, is not used to select the Ethernet controller chip, but it must be kept as an output or the SPI interface won't work.  In the Arduino code we would need to include the library as follows: #include <SPI.h> #include <Ethernet.h>  https://www.arduino.cc/en/uploads/Reference/arduino_uno_ethernet_pins.png  https://www.arduino.cc/en/uploads/Reference/arduino_mega_ethernet_pins.png  Ethernet Module  The Arduino Ethernet Shield allows you to easily connect your Arduino to the internet or communicate to other modules in the Ethernet bus. The Ethernet Shield is based upon the W51000 chip, which has an internal 16K buffer. It has a connection speed of up to 10/100Mb. It relies on the Arduino Ethernet library, which comes bundled with the development environment.    The Arduino board can be interfaced to the Ethernet module as shown.   |  |  | | --- | --- | |  |  |   Plug the Arduino Uno board into your computer's USB port, and the Ethernet shield into your router (or direct internet connection).    **Arduino IDE Software**  Open Arduino IDE software and to find the IP address that has been assigned to the Ethernet board:  Open the DhcpAddressPrinter sketch..  File --> Examples --> Ethernet --> DhcpAddressPrinter  On new Ethernet boards the address is located on a sticker attached to the board. If there is no address a unique MAC address can be used. If multiple Ethernet modules are present each board may be assigned a unique address. Once the MAC address is properly configured, upload the sketch to the Arduino board. Open the serial monitor. This will print out the IP address in use.  **Program 1: Client Server**  **Client Code**  //Client  #include <Ethernet.h>  #include <SPI.h>  byte mac[] = {0xDE,0xAD,0xBE,0xEF,0xFE,0xEC};  IPAddress ip(192,168,1,177);  IPAddress server(192,168,1,180);  EthernetClient client(10001);  void setup()  {  Ethernet.begin(mac, ip);  Serial.begin(9600);  delay(1000);  Serial.println("connecting...");  if (client.connect(server, 10001)) {  Serial.println("connected");  client.println("GET /search?q=arduino HTTP/1.0");  client.println();  } else {  Serial.println("connection failed");  }  }  void loop()  {  if (client.available()) {  char c = client.read();  Serial.print(c);  }  if (!client.connected()) {  Serial.println();  Serial.println("disconnecting.");  client.stop();  for(;;)  ;  }  }  **Server Code**  //Server  #include <SPI.h>  #include <Ethernet.h>  // network configuration. gateway and subnet are optional.  // the media access control (ethernet hardware) address for the shield:  byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };  //the IP address for the shield:  byte ip[] = { 192,168,1,180 };  // the router's gateway address:  byte gateway[] = { 192, 168, 1, 1 };  // the subnet:  byte subnet[] = { 255, 255, 255, 0 };  EthernetServer server = EthernetServer(10001);  void setup()  {  // initialize the ethernet device  Ethernet.begin(mac, ip, gateway, subnet);  // start listening for clients  server.begin();  }  void loop()  {  // if an incoming client connects, there will be bytes available to read:  EthernetClient client = server.available();  if (client == true) {  // read bytes from the incoming client and write them back  // to any clients connected to the server:  server.write(client.read());  }  } |
|  |

**Program 2**

**CAN communication**



|  |  |
| --- | --- |
|  |  |

**CAN Transmitter Code (Arduino Nano)**

#include <SPI.h>          //Library for using SPI Communication

#include <mcp2515.h>      //Library for using CAN Communication

#include <DHT.h>          //Library for using DHT sensor

#define DHTPIN A0

#define DHTTYPE DHT11

struct can\_frame canMsg;

MCP2515 mcp2515(10);

DHT dht(DHTPIN, DHTTYPE);     //initilize object dht for class DHT with DHT pin with STM32 and DHT type as DHT11

void setup()   
{  
  while (!Serial);  
  Serial.begin(9600);  
   SPI.begin();               //Begins SPI communication  
  dht.begin();               //Begins to read temperature & humidity sensor value  
  mcp2515.reset();  
  mcp2515.setBitrate(CAN\_500KBPS,MCP\_8MHZ); //Sets CAN at speed 500KBPS and Clock 8MHz  
  mcp2515.setNormalMode();  
}

void loop()   
{  
  int h = dht.readHumidity();       //Gets Humidity value  
  int t = dht.readTemperature();    //Gets Temperature value

  canMsg.can\_id  = 0x036;           //CAN id as 0x036  
  canMsg.can\_dlc = 8;               //CAN data length as 8  
  canMsg.data[0] = h;               //Update humidity value in [0]  
  canMsg.data[1] = t;               //Update temperature value in [1]  
  canMsg.data[2] = 0x00;            //Rest all with 0  
  canMsg.data[3] = 0x00;  
  canMsg.data[4] = 0x00;  
  canMsg.data[5] = 0x00;  
  canMsg.data[6] = 0x00;  
  canMsg.data[7] = 0x00;  
  mcp2515.sendMessage(&canMsg);     //Sends the CAN message  
  delay(1000);  
}

**CAN Receiver Code (Arduino UNO):**

#include <SPI.h>              //Library for using SPI Communication   
#include <mcp2515.h>          //Library for using CAN Communication  
#include <LiquidCrystal.h>    //Library for using LCD display

const int rs = 3, en = 4, d4 = 5, d5 = 6, d6 = 7, d7 = 8;

LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

//Define LCD display pins RS,E,D4,D5,D6,D7

struct can\_frame canMsg;   
MCP2515 mcp2515(10);                 // SPI CS Pin 10   
   
 void setup() {  
  lcd.begin(16,2);                   //Sets LCD as 16x2 type  
  lcd.setCursor(0,0);                //Display Welcome Message  
  lcd.print("BITS-PILANI");  
  lcd.setCursor(0,1);  
  lcd.print("CAN ARDUINO");  
  delay(3000);  
  lcd.clear();  
    
  SPI.begin();                       //Begins SPI communication  
  Serial.begin(9600);  //Begins Serial Communication at 9600 baudrate   
  mcp2515.reset();

//Sets CAN at speed 500KBPS and Clock 8MHz                   mcp2515.setBitrate(CAN\_500KBPS,MCP\_8MHZ);   
  mcp2515.setNormalMode();                  //Sets CAN at normal mode  
}

void loop()   
{  
  if (mcp2515.readMessage(&canMsg) == MCP2515::ERROR\_OK)

// To receive data (Poll Read)  
  {  
     int x = canMsg.data[0];           
     int y = canMsg.data[1];         
        
      lcd.setCursor(0,0);//Display Temp & Humidity value received at 16x2 LCD  
      lcd.print("Humidity : ");  
      lcd.print(x);  
      lcd.setCursor(0,1);  
      lcd.print("Temp : ");  
      lcd.print(y);  
      delay(1000);  
      lcd.clear();  
    }  
}

**Exercise**

The J1939 protocol document provides the following information for Battery voltage measurement.

SPN: 168

PGN: 65271

CAN ID: 0xFEF7

Byte Position: 5-6

Resolution: 0.05V/bit

Offset: 0

Min Data: 0

Max: 3212.75

Units: Voltage

Gauge: 0-32

Assume that a potentiometer is connected at the analog input pin of Board 1

to input a voltage in the range of 0 to5V. Send the data of this voltage in a 8-byte frame so that the voltage can be expressed as a 16-bit value occupying bytes 5 and 6 with the byte 5 being the lowest significant byte.

The data should be read by CAN board 2 and displayed in a LCD in the range of 0 to 48V.

Write programs for the following

1. Board 1 should read the potentiometer connected to analog input pin, perform an ADC conversion and load the value as two bytes in Byte 5 and Byte 6 in a 8-byte array. Send the data using CAN protocol with CAN ID of 0xFEF7 (alternatively use a 11 bit ID).

Board 2 should read this array of data, convert the data from Bytes 6 and Byte 5 combine them scale them to a value from 0 to 48V and display the value to an LCD. If the potentiometer connected to Board 1 changes from 0 to 5V, the value in the LCD will change from 0 to 48V approximately.

Notes: Reference the Arduino Examples on how to read analog value and perform analog to digital conversion