PROYECTO
DE COMUNICACIONES INDUSTRIALES

Gabriel Cuesta (cuesta.147993) y Hugo Collantes (collantes.147173)
INGENIERIA INDUSTRIAL

Índice

Introducción

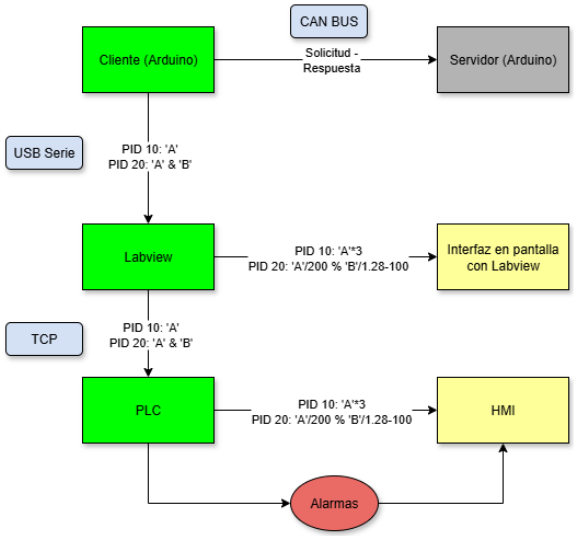
El objetivo de este proyecto es el **desarrollo de una red industrial que simule la comunicación de un vehículo**. Para ello, se emplearán una comunicación a través de Arduino, simulándola mediante dos nodos, cliente y servidor. Para que el proyecto sea una simulación lo más cercana a la realidad, se centra en 2 PIDs que representan parámetros concretos del vehículo.

Este sistema de comunicación con PIDs se conocen como **OBDII**, un método muy usual en la comunicación de vehículos. A nuestro grupo se le asignan los **PID 10 y 20**. El PID 10, de 1 byte, representa la **presión del combustible** medido en kPa y el PID 20, de 2 bytes, representa el **sensor de oxígeno 1**, que emplea 1 byte para indicar la tensión del sensor y el otro byte para indicar el porcentaje medido.

0A	10	1	Fuel pressure (gauge pressure)	0	765	kPa	$3A$
14	20	2	Oxygen Sensor 1 A: Voltage B: Short term fuel trim	0 -100	1.275 99.2	V %	$\frac{A}{200}$ $\frac{100}{128}B - 100$

A la derecha se aprecian las fórmulas correspondientes. De forma que el **primer byte es 'A'** y el **segundo byte es 'B'**. Para obtener el valor real se aplica la correspondiente fórmula para obtener un valor entre los límites impuestos.

Una vez entendido como funciona el PID, ya se puede definir el flujo del proyecto. Como se puede observar en el diagrama, se comienza con la comunicación entre **Servidor Y Cliente**, en Arduino, empleando **CAN BUS**. A continuación, se emplea comunicación USB Serie para enviar la información al ordenador, donde se procesa con **Labview**. En Labview se procesa y opera para mostrarlo en su propia interfaz y se envía por medio de **TCP al PLC**. Una vez en el PLC se procesa y opera para mostrarlo en su correspondiente **HMI**. Es en esta etapa donde se añaden las **alarmas al sistema**.



Hay que recalcar que en todo momento la comunicación se efectúa con los bytes ‘A’ y ‘B’, no con sus valores reales, esto se hace para **mantener la máxima precisión** en la información, será en LabVIEW y en el PLC donde se operará.

Con esto ya queda definido el proyecto, a continuación, se explicará con detalle cada apartado.

Can Bus

Se dispone de dos códigos diferentes en Arduino para la comunicación de ambos nodos. El código del cliente es el siguiente:

```
CAN.sendMsgBuf(DIR_OBDII, 0, canLen, stmp);

 if(flagRecv)
 {                                   // comprobamos si esta variable vale 1. Si vale 1 es que hemos recibido datos

    flagRecv = 0;                    // borramos esta variable o flag

    CAN.readMsgBuf(&len_Recv, buf_Recv); // read data,  len: data length, buf: data buf
 // Procesado de datos OBD
  obdLen = buf_Recv[0];
  obdService = buf_Recv[1];
  obdPID = buf_Recv[2];

 Serial.print(obdPID, DEC);
 Serial.print(";");
 if (obdPID==PID_2){
 Serial.print(buf_Recv[3], DEC);
 Serial.print(",");
 Serial.println(buf_Recv[4], DEC);
 }
 if (obdPID==PID_1){
    Serial.println(buf_Recv[3], DEC);
 }
 }
delay(1000);
 }
```

El cliente es quien **solicita información al servidor y procesa la repuesta**. En este fragmento de código se ve como interpretará los bytes que recibe el cliente, el primero, representará la longitud de la trama restante. El segundo será el modo utilizado, el modo 1 en este caso. El tercer byte, representará al PID en cuestión y los bytes restantes representarán la información de dicho PID. En este caso se evaluarán los PID 10 y 20. El primero de ellos será el PID_1 del código con un byte de información y el PID_2 representará al segundo PID, con dos bytes de información.

En cuanto al servidor, que se encarga de enviar la trama de datos solicitados por parte del cliente, implementa el siguiente código.

Donde comienza con la definición de la lista de datos, todos estimados como valores reales. Se observan 3 listas: la primera es el byte del PID10 y las otras dos los bytes A, B del PID 20.

```
5   #define DIR_CLIENT 0x7E8 // Dirección del Cliente (Respuesta del servidor/ECU)
6   #define DIR_SERVER_LISTEN 0x7DF // Dirección de escucha del servidor (Petición del cliente)
7
8   const int SPI_CS_PIN = 9;
9   MCP_CAN CAN(SPI_CS_PIN);
10
11  unsigned char valor_fuel[] = {90, 111, 96, 103, 100, 118, 130, 125, 108, 99};
12  unsigned char valor_oxigen1[] = {20, 40, 60, 80, 100, 110, 120, 140, 160, 180 };
13  unsigned char valor_oxigen2[] = {116,118,120,122,124,126,128,130,134,140};
14
```

Se establece la correcta interpretación por parte del servidor ante la **trama del cliente** estableciendo el mismo criterio que en el cliente.

```
// Leer el mensaje CAN entrante
// Pasamos &len_Recv y buf_Recv como punteros a la funcion
CAN.readMsgBuf(&len_Recv, buf_Recv);

// Siempre que recibamos algo, lo procesamos. No necesitamos 'if (len_Recv > 0)' aquí dentro.

// Procesado de datos OBD de la PETICIÓN ENTRANTE
obdLen = buf_Recv[0];
obdService = buf_Recv[1];
obdPID = buf_Recv[2];
```

Se ve a continuación como se van enviando datos según el PID solicitado por el cliente, la lista de datos va avanzando hasta alcanzar su límite donde se reinicia. Para cada PID se hace de forma diferente enviando uno o dos bytes.

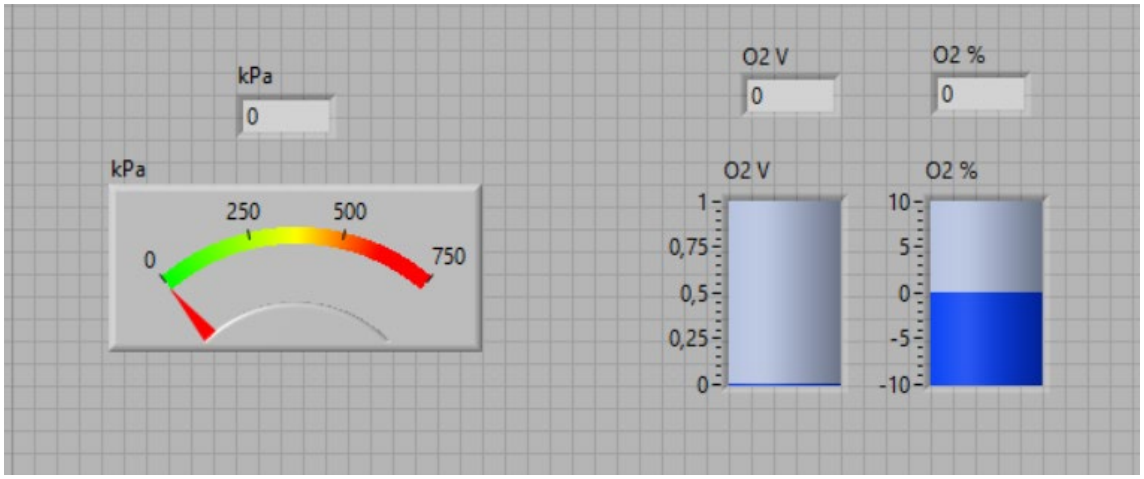
```
if (obdPID == 10){
  stmp[3] = valor_fuel[indice_fuel]; // Dato A (Byte 3 de la trama CAN)
  obdLen = 3; // La longitud de datos utiles que declaramos en el byte 0 de la respuesta
  indice_fuel++;
  if (indice_fuel > 9){ indice_fuel = 0; }
} else if (obdPID == 20){
  stmp[3]= valor_oxigen1[indice_oxigen1]; // Dato A
  stmp[4] = valor_oxigen2[indice_oxigen2]; // Dato B
  obdLen = 4; // Longitud de datos utiles
  indice_oxigen1++;
  indice_oxigen2++;
  if (indice_oxigen1>9){
      indice_oxigen1=0;
      indice_oxigen2=0;
  }
} else {
  // PID no reconocido, no respondemos
  Serial.print("Parámetro ");
  Serial.print(obdPID, DEC);
  Serial.println(" no contemplado en el servidor. No se envia respuesta.");
  doResponse = false;
}
```

Se establece una respuesta al recibir un PID diferente a estos ya que no supone algo relevante para este proyecto.

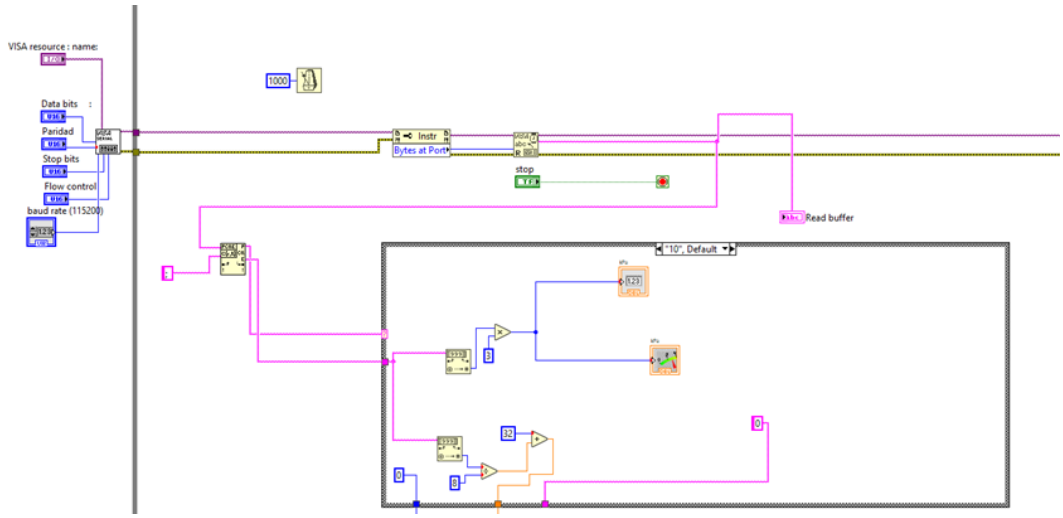
Finalmente, hay que recalcar que la información recibida por parte del cliente no será la implementada ya con las fórmulas. Estas se implementarán en Labview y el PLC donde se mostrarán los **valores reales de forma más visual**.

Labview (USB Serie)

Una vez realizada la comunicación entre ambos nodos, se introduce dentro de Labview. Donde se usa un bloque “match pattern” con el fin de s**eparar tanto el número del PID, como de la información que contiene**. Se introduce en un bloque “case” donde diferenciamos el número del PID para el caso del “10” y del “20”. Se observa como para el caso del PID 10 se implementa el cálculo ya real, que es multiplicar por tres y se muestra en un elemento visual.



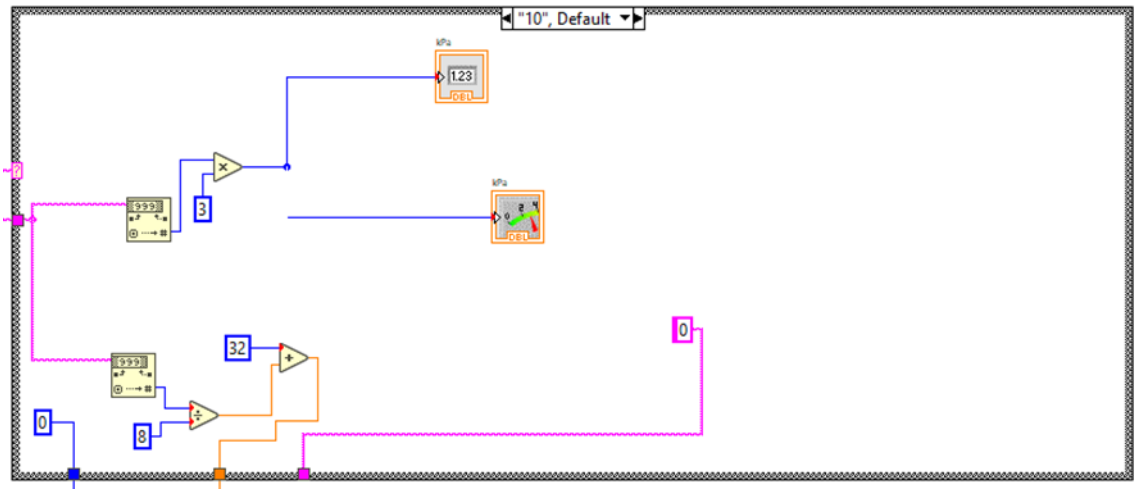
Para trabajar con la información uno de los pasos clave es la conversión de los valores, queremos operar los bytes ‘A’ y ‘B’ pero en formato “string” esto nos es posible, para ello se convierte el valor a un “int”.



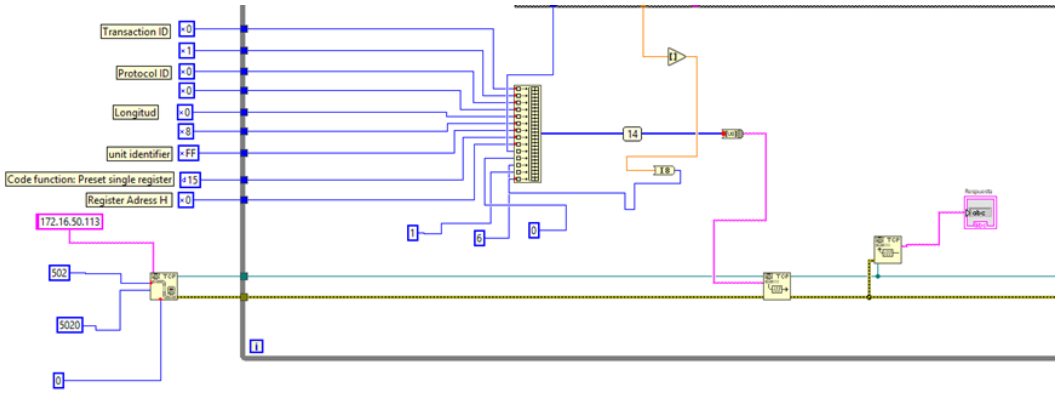
Pese a no mostrar cómo se implementó el PID 20, no tiene ningún misterio es exactamente igual, pero variando tanto las fórmulas para mostrarlo de forma real.

Modbus TCP/IP

Para implementar la comunicación Modbus TC/IP se requiere adaptar la información antes de ser enviada para el PLC para ello se utiliza la estructura observada a continuación:



Hay que remarcar que de ambos bytes solo se utilizará el A para mostrar y ser enviado a través de la conexión TCP/IP. Al existir dos opciones de mostrar dicha información ya sea en **voltaje** (byte A) como en **%** (byte B), se ha escogido el **voltaje**. En caso de disponer de mas tiempo para el proyecto, se podrían haber buscado otras funciones que permitieran la comunicación del byte 'B' al PLC.



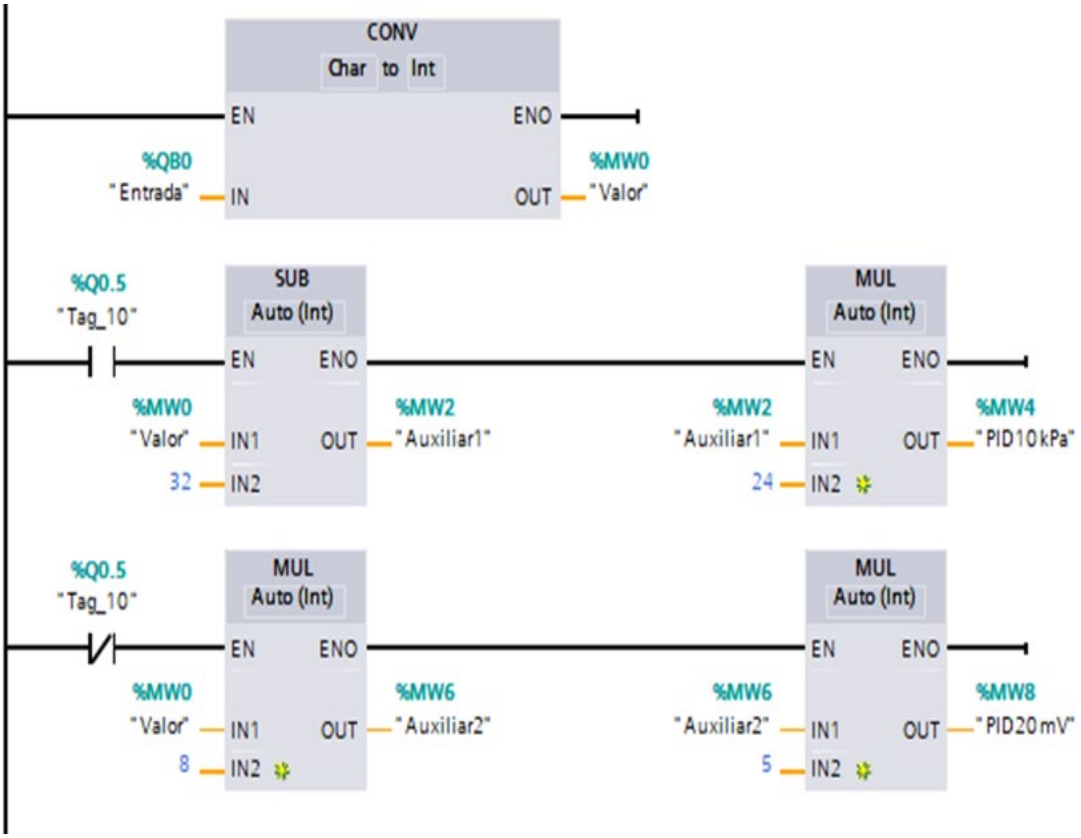
La función escogida es la 15; "Force multiple Coilds (Outputs)", con ella podemos enviar, **teóricamente hasta 1 byte** que se mostrara en las salidas digitales, por limitaciones del PLC solo se podrán emplear **6 salidas, esto limita la precisión**. En estos 6 bits necesitamos enviar la información y diferenciar el PLC, para ello se divide el valor del byte 'A' entre 8, **pasando de 8 bits a 5** y se añade usa el **bit 6 como diferenciador**, de modo que el mensaje del **PID 10** tendrá este bit en **alto** (mediante la suma de 32) y el **PID 20** siempre en **bajo.**

Esta parte del proceso consiste en generar un **Array de valores** que representen los **distintos campos de la comunicación TCP**, como son la función empleada, el espacio al que va dirigido, la longitud del mensaje, etc.

Finalmente, este mensaje TCP contiene los valores del byte 'A' de los 2 PIDs, escalados a 5 bits y usando el 6º bit para diferenciar el PLC. Este 6º bit deberá tenerse en cuenta en el operado del PLC.

HMI

Una vez recibidos los datos, a través de TIA Portal, se comienza con una conversión del valor a **formato "int"** para trabajar con mayor facilidad. Se establecen dos estructuras, diferentes para mostrar la información en el HMI. Cada una de ellas se activará en función del 6 bit recibido que es el que diferencia ambos PID.

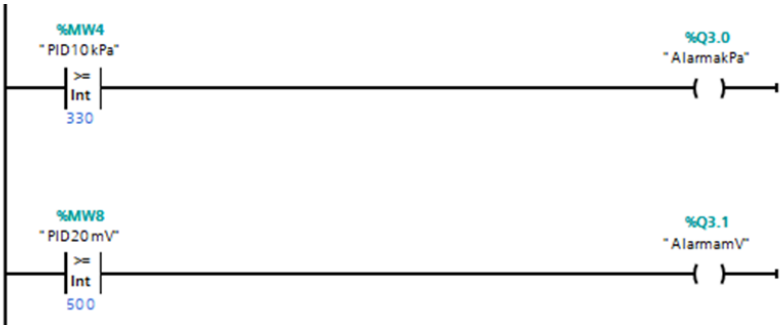


En cuanto al PID 10 que es la primera estructura, se ve como nos llega el valor puro (no es el valor real, con la fórmula implementada) y se le **resta 32** que se ha añadido anteriormente. La variable auxiliar entra en un bloque de multiplicación por 24. Esto se hace x8 ya que anteriormente lo reducimos de 8 a 5 bits y x3 ya que se representa así valor real en KPa del PID 10.

Para el PID 20 multiplicamos por 8 por el mismo motivo y además por 5. Esto proviene de que la fórmula $valor\ real\ (V) = \frac{(valor\ puro)}{200}$ y al quererlo mostrar en mV multiplicamos por 1000, de ahí se obtiene dicho factor.

Alarmas

Se establecen diferentes umbrales para ambos PID. El límite inferior para el **PID 10** será **330 kPa**, y para el **PID 20** será de **500 mV**. Dichos valores han sido considerados teniendo en cuenta los valores limites y habituales de los PIDs.



31	AlarmakPa	Standard-Variablen...	Bool	%Q3.0	☐	☑	☑
32	AlarmamV	Standard-Variablen...	Bool	%Q3.1	☐	☑	☑

Resultado Final

Finalmente se muestra la interfaz final del proyecto, donde se observan **los valores de manera visual y numérica** y se añaden **LEDs que representan las alarmas**. Esto permite observar la información de los PIDs de manera directa y visual.

[Video del Resultado]