

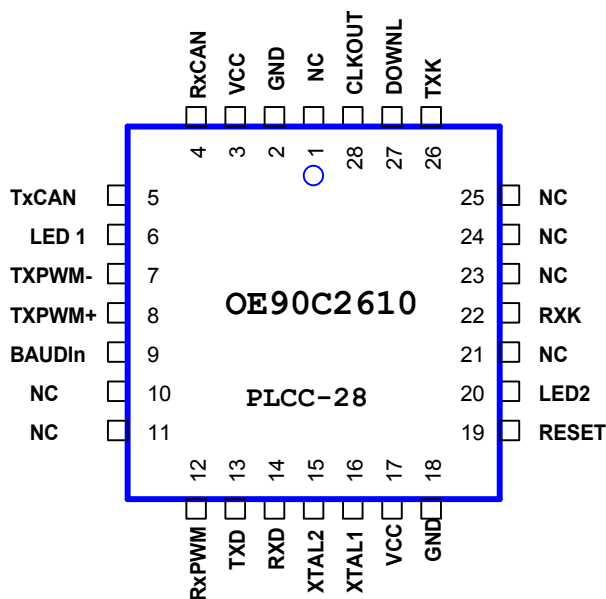


Features

- Compatible with FORD motors
- RS232 communication at 9600 Baud
- Extended codes
- FORD ISO9141 support
- FORD SCP support
- FORD CAN support
- Access to Airbag,ABS,Dashboard etc.
- Read and clear manufacturer DTCs
- Hardware compatible with mobydic2600

Description

OE90C2610 is specially designed for FORD motor company cars. This chip allows to build a complete diagnostic system using a PC or directly in small device with LCD display. Some other brand like Mazda , Jaguar , Aston Martin , Volvo can be tested. All diagnostic mode are implemented . Some commands are implemented to allow direct access to different ECUs. This chip is the best way with a appropriated software to reflash the ECU. This chip is plug play compatible with mobydic2600 Hardware and PCB.



ÖZEN
ELEKTRONIK

FORD to RS232 gateway

OE90C2610

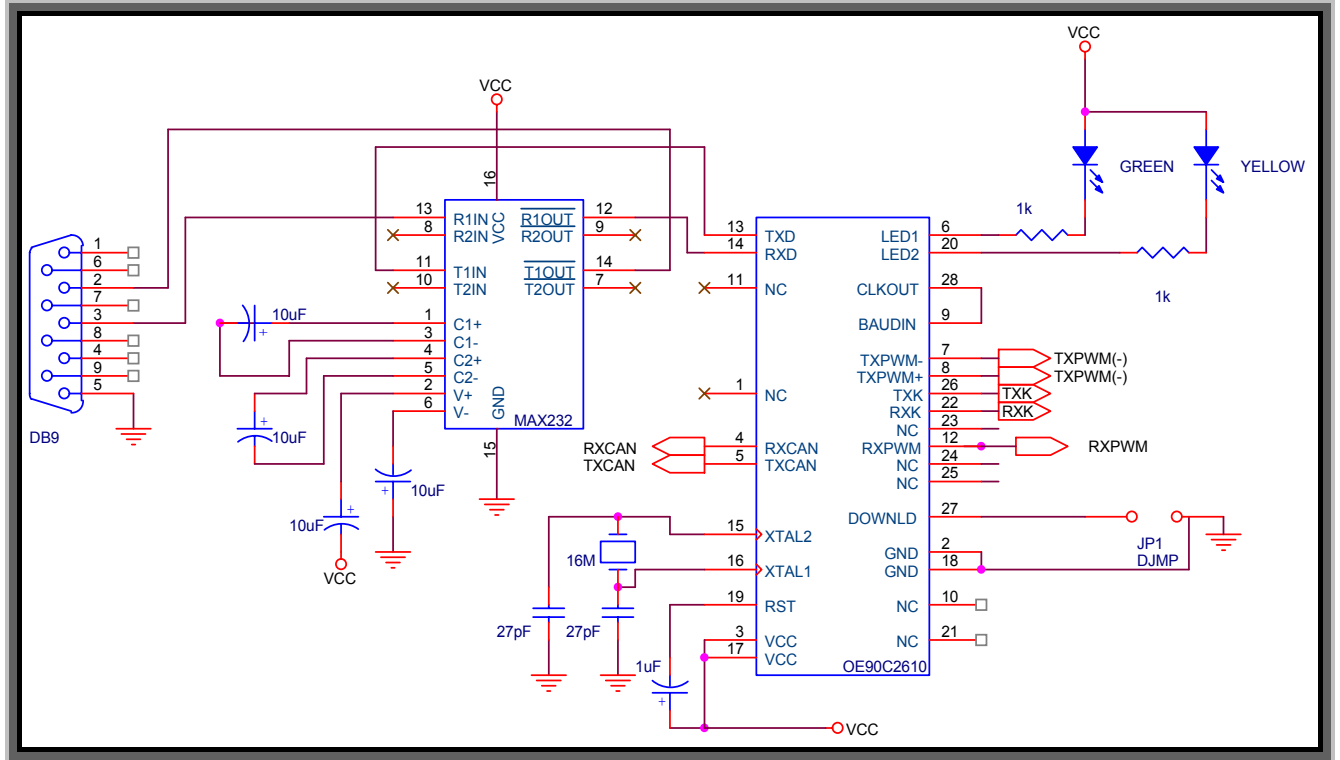


Pin description

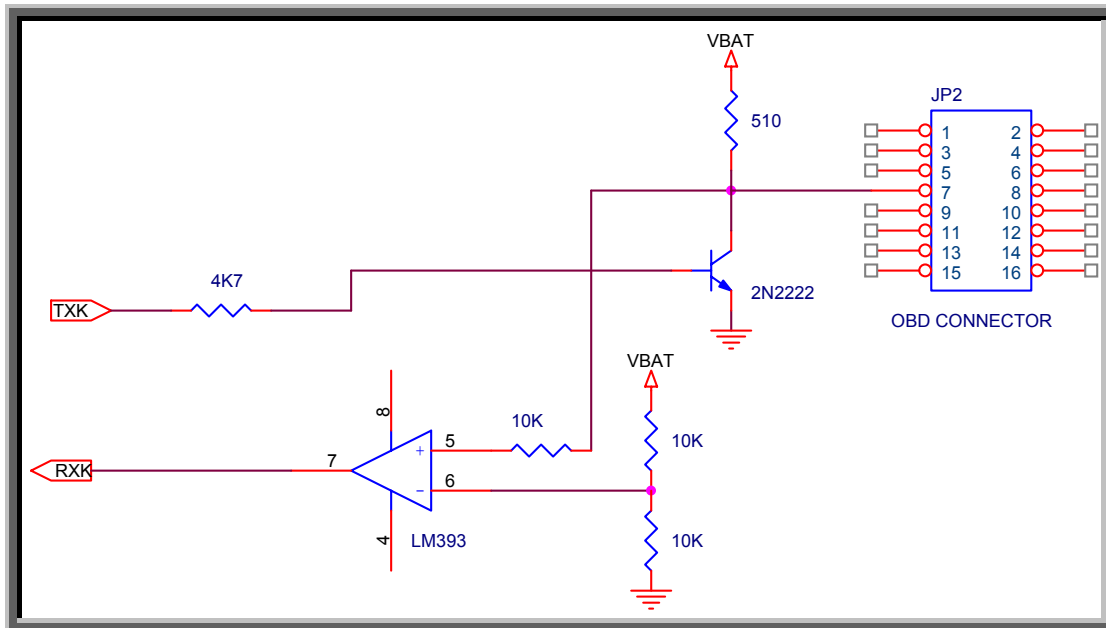
Pin	Pin Name	Type	Description
1	NC		
2	GND		Ground
3	VCC		Supply voltage
4	RXCAN	I	FORD CAN BUS input
5	TXCAN	O	FORD CAN BUS output
6	LED1	O	OK LED max 5 mA for low current LED
7	TXPWM-	O	Transmit (-) output for SCP signal
8	TXPWM+	O	Transmit (+) output for SCP signal
9	BAUDIN		RS232 Baudrate input clock
10	NC		
11	NC		
12	RXPWM	I	FORD SCP input
13	TXD	O	RS232 output
14	RXD	I	RS232 input
15	XTAL2	I	16 Mhz crystal input
16	XTAL1	I	16 MHz crystal input
17	VCC		Supply voltage
18	GND	I	Ground
19	RESET	I/O	A high level on this pin during 2 machine cycles while the oscillator is running resets the device.
20	LED2	O	LED output to indicate the frames exchange
21	NC		
22	RXK	I	FORD ISO9141 K-line input
23	NC		
24	NC		
25	NC		
26	TxK	O	FORD ISO9141 K-Line output
27	DOWNLD	I	A low on this pin puts the device in download mode
28	CLKOUT	O	Clock output for RS232 baud rate in



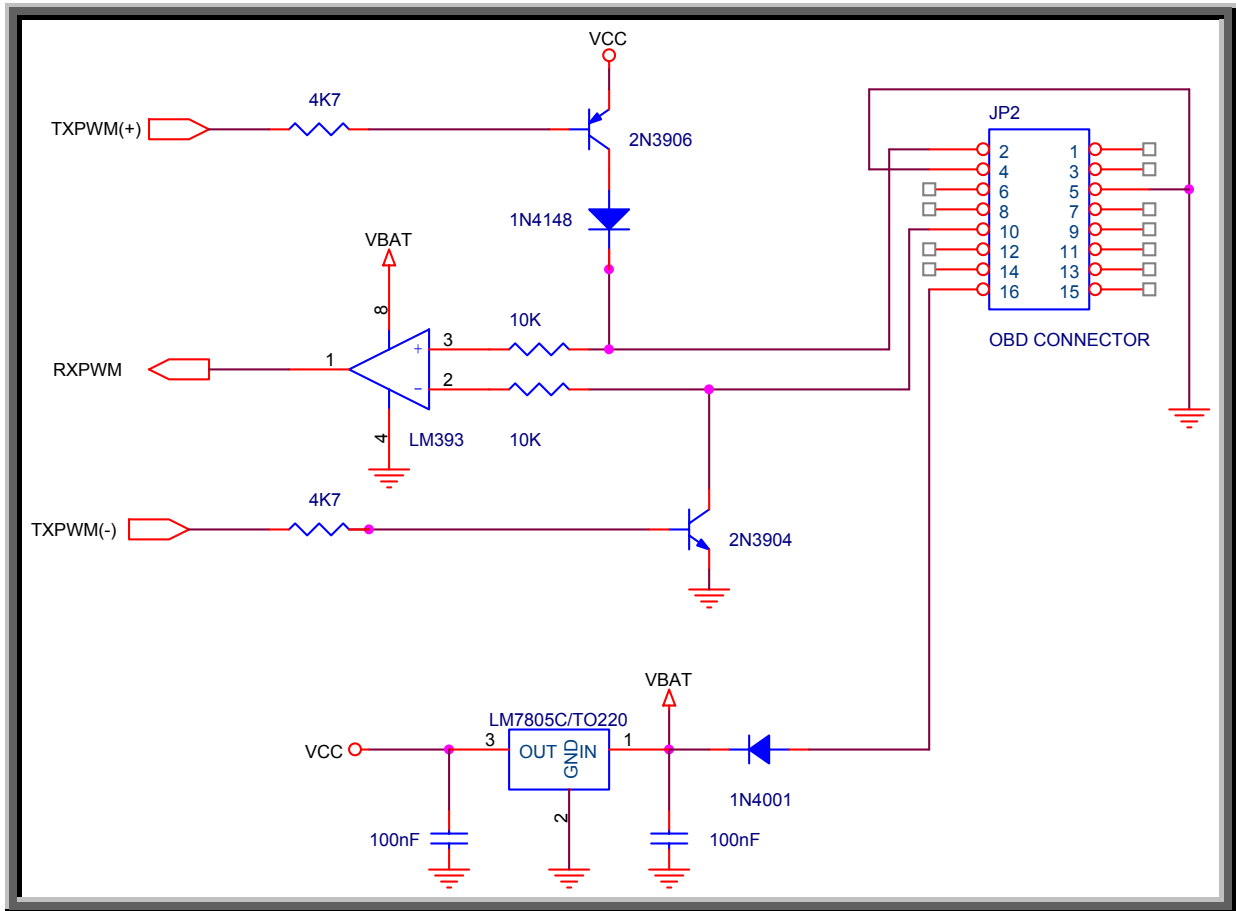
HARDWARE INTERFACE



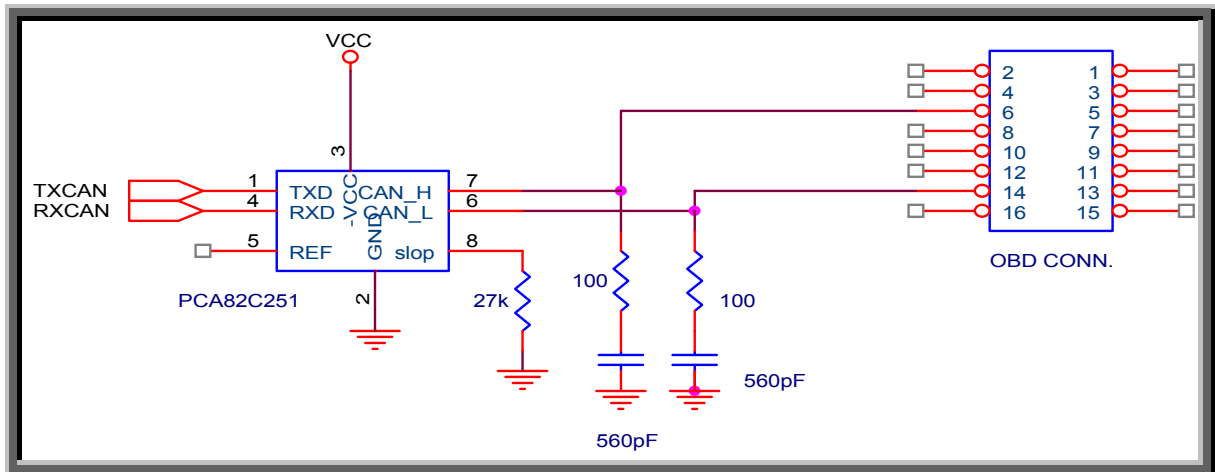
Typical connection of OE90C2610



Typical FORD-9141 Interface



Typical FORD SCP Interface & Power supply



Typical FORD-CAN Interface



FORD PROTOCOL

The chip OE90C2610 allows the implementation of Serial communication Protocols used on FORD vehicles. These Communication Protocols are:

- FORD-9141 VEHICLE DIAGNOSTIC LINK
- STANDARD CORPORATE PROTOCOL (SCP)
- FORD-CONTROLLER AREA NETWORK (FORD-CAN)

Ford9141 is a single wire media and meets all requirement of FORD-9141 protocol definition and Interface requirement document. It support ABS , Airbag , Parking aid Module , Security modul.....and its baudrate is 10.4 KB.

The SCP physical layer is based on SAE J1850-PWM protocol at 41.6 KB , the dashboard and PCM/TCM are connected via this BUS a IFR type 1 is required.

The FORD CAN is in compliance with ISO11898 and with SAE J2284 for vehicle application at 500 kB.

This chip can allow the communication with another brands using the FORD motors. (Land Rover , Mazda , Aston Martin etc...)



SOFTWARE INTERFACING TO CHIP

OE90C2610 communicates via RS232 with 9600/8/n/1 parameter . No Software Handshake is necessary prior to send a message .

PC is master and mOByDic is slave. The communication works on half duplex basis. The 2610 has a communication timeout of 3 sec.

RS232 device driver routines

If the user sends a 1 to mOByDic , mOByDic responds with 1 . This function allows a line checking or device checking. Some delphi exemples show how to begin to communicate with mOByDic. Please note that the following routines dont intend to be used in a professional application , they only allow the user to understand the messaging strategy of mOByDic 2610.

```
unit mOByDic_device_driver;  
  
interface  
  
uses  
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,  
  Dialogs, StdCtrls;  
  
var  
  
  hCommFile      : THandle;  
  DCB             : TDCB;           // structure for rs232  
  read_Error     : boolean;       // RS232 Read error by timeout  
  RS232_Buffer   : array [0..255] of byte; // buffer for incoming bytes
```

implementation



```

/*****
/**
/** Function: open and initialize COM1 port 9600 baud / 8 / n / 1
/** Input : -----
/** Output :
/**
/*****

Procedure Open_COMM;
begin
    // change COM1 if another port will be used

    hCommFile := CreateFile('COM1', GENERIC_READ or GENERIC_WRITE,
        0, Nil, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);

    GetCommState(hCommFile,DCB); // set comm parameter
    with DCB do
        begin
            Baudrate:= CBR_9600;
            ByteSize:= 8;
            Parity := NOPARITY;
        end;
    SetCommState(hCommFile,DCB);

    if hCommFile = INVALID_HANDLE_VALUE then // error at open port
        begin
            ShowMessage ('ERROR OPENING PORT');
            exit;
        end;

end;

/*****
/**
/** Function: close a open comm port
/** Input :
/** Output :
/**
/*****

Procedure close_COMM;
begin
    closeHandle(hCommFile);
end;

/*****
/**
/** Function: clearBuffers, clear rs232 Tx and Rx buffers
/** Input :
/** Output :
/**
/*****

Procedure clearBuffers;
begin
    purgecomm(hcommfile,PURGE_TXCLEAR);
    purgecomm(hcommfile,PURGE_RXCLEAR);
end;

/*****
/**
/** Function: write a byte to open RS232 port no timeout occurs
/** Input : tosend : byte to send
/** Output :
/**
/*****

Procedure write_RS232 ( tosend:byte);
```



```

Var sent:dword;
begin
  WriteFile (hCommFile,tosend,1,sent,nil);
end;

/*****
/*
/* Function: Read a byte from RS232
/* Input : Timeout value in mS to wait on first byte
/* Output : received char , Read_error flag is true if any error
/*
/*
/*****

Function read_RS232(Timeout_value:word) : byte;
var
  TimeOutBuffer:PCOMMTIMEOUTS;
  bytesread:dword;
  read_byte:byte;

begin
  //timeout parameter setting

  GetMem(TimeOutBuffer,sizeof(COMMTIMEOUTS));
  GetCommTimeouts (hCommFile,TimeOutBuffer^);
  TimeOutBuffer.ReadTotalTimeoutMultiplier:=0;
  TimeOutBuffer.ReadTotalTimeoutConstant:=timeout_value;
  SetCommTimeouts (hCommFile,TimeOutBuffer^);

  ReadFile(hCommFile,Read_byte,1,bytesread,nil);
  if (BytesRead = 0)
    then begin
      Read_RS232:= 0;
      read_error := true;
    end
    else begin
      read_error := false;
      Read_RS232:= read_byte;
    end;
end;

end.

```

MOByDic 2610 Command list

Commande	01
Fonction	First contact to 2610 / line checking
Request	01
Pos. Response	01
Neg. Response	N/A



Exemple for command 01

```

Open_COMM;
clearbuffers;
write_rs232(1);
rs232_buffer[0]:=read_rs232(100); // send first contact message
// get the response with 100 ms timeout
  if read_error
  then ..... // no contact
  else if rs232_buffer[0]=1
  then ..... // first contact OK device existes
  else ..... // no contact
    
```

Commande	03
Fonction	Read serial number of 2610
Request	03
Pos. Response	03 HB LB HB = high byte of serial number LB = low byte of serial number Serial# = (256*HB) + LB Actually not used and give 00 00 back
Neg. Response	N/A

Exemple for command 03

```

Open_COMM;
clearbuffers;
write_rs232(3); // send read serial number command
rs232_buffer[0]:=read_rs232(100); // get the response with 100 ms timeout
// no response
if read_error then exit
rs232_buffer[1]:=read_rs232(10); // get the response with 10 ms timeout
// no response
if read_error then exit
rs232_buffer[2]:=read_rs232(10); // get the response with 10 ms timeout
// no response
if read_error then exit
Serial:=(256*rs232_buffer[1])+rs232_buffer[2]; // get serial
    
```

Commande	04
Fonction	Read version of 2610
Request	04
Pos. Response	04 HB LB HB = high byte version LB = low byte of version Version# = (256*HB) + LB (220 decimal => ver 2.20)
Neg. Response	N/A



Exemple for command 04

```

Open_COMM;
clearbuffers;
write_rs232(4); // send read version number command
rs232_buffer[0]:=read_rs232(100); // get the response with 100 ms timeout
if read_error then exit // no response
rs232_buffer[1]:=read_rs232(10); // get the response with 10 ms timeout
if read_error then exit // no response
rs232_buffer[2]:=read_rs232(10); // get the response with 10 ms timeout
if read_error then exit // no response
Version:=(256*rs232_buffer[1])+rs232_buffer[2]; // get version in x100 format
    
```

Commande	05
Fonction	Soft Reset of 2610
Request	05
Pos. Response	ACK (06)
Neg. Response	N/A

Exemple for command 05

```

Open_COMM;
clearbuffers;
write_rs232(5); // send soft reset command
rs232_buffer[0]:=read_rs232(100); // get the response with 100 ms timeout
if read_error then exit // no response
.....
    
```

Commande	06
Fonction	Read chip ident of mOByDic
Request	06
Pos. Response	06 HB LB HB = high byte of chip ident LB = low byte of chip ident Version# = (256*HB) + LB 2610 = OE90C2610
Neg. Response	N/A

Exemple for command 06



```

Open_COMM;
clearbuffers;
write_rs232(6); // send read chip ident command
rs232_buffer[0]:=read_rs232(100); // get the response with 100 ms timeout
if read_error then exit // no response
rs232_buffer[1]:=read_rs232(10); // get the response with 10 ms timeout
if read_error then exit // no response
rs232_buffer[2]:=read_rs232(10); // get the response with 10 ms timeout
if read_error then exit // no response
chip_ident:=(256*rs232_buffer[1])+rs232_buffer[2]; // get chip ident
    
```

Commande	10
Fonction	Set ECU Address on FORD9141 BUS
Remarque	Default setting is \$28 (ABS)
Request	10 , ECU_Address
Pos. Response	ACK (06)
Neg. Response	N/A

Exemple command 10 how setting the header in mObydic to ABS via ISO9141 Bus

```

Open_COMM;
clearbuffers;
write_rs232(10); // send set address command
write_rs232($28); // send ABS address to mOByDic
rs232_buffer[0]:=read_rs232(100); // get the response with 100 ms timeout
if read_error then exit // no response
if rs232_buffer[0] = ACK
    then ..... // handle
    
```



Exemple for command 10 how setting the header to Airbag Modul via ISO9141 Bus

```

Open_COMM;
clearbuffers;
write_rs232(10); // send set address command
write_rs232($58); // send Airbag address to mOByDic
rs232_buffer[0]:=read_rs232(100); // get the response with 100 ms timeout
if read_error then exit // no response
if rs232_buffer[0] = ACK
    then ..... // handle
    
```

Commande	11
Fonction	Send a free message to Ford9141 BUS
Remarque	This message is sent to the Modul pointed by command 10 User needs only to send the bytes in data field of ford message format. No header sending is necessary. MOByDic uses already existing header.
Request	11 , data_length , <data>
Pos. Response	11 , length , Header1 , 0xF1 , Ecu_address , <data> , CS
Neg. Response	Data_length byte doesnt match the data length

Exemple command 11 how clear DTC in by command 10 pointed modul

FIRST POINT TO MODUL AIRBAG

```

Open_COMM;
clearbuffers;
write_rs232(10); // send set address command
    
```



```
write_rs232($58); // send Airbag address to mOByDic
rs232_buffer[0]:=read_rs232(100); // get the response with 100 ms timeout
if read_error then exit // no response
if rs232_buffer[0] <> ACK then exit;
```

REQUEST CLEAR DTC FROM MODUL AIRBAG USING COMMAND 11

```
write_rs232(11); // send MESSAGE command
write_rs232(3); // 3 Bytes follow
write_rs232($14); // 14 is clear DTC code
write_rs232($ff); // FF
write_rs232(00); // 00 all group

rs232_buffer[0]:=read_rs232(250); // wait for 1.byte
if read_error then exit // no response
if rs232_buffer[0] <> 11 the exit // false sync.
Length := read_rs232(10); // get incoming message length
For i:=1 to length do
  Begin
    rs232_buffer[i]:=read_rs232(10); // read length x byte 10 ms interbyte timeout
  End;
```

In Rs232_buffer user has following data :

11 , 07 , 64 , F1 , 58 , 54 , FF , 00 , CS

11 : response to command 11 (echo)
 07 : total bytes follow
 64 : type of ford9141 message header
 F1 : targeted address (mobydic)
 58 : source address (airbag)
 54 : response to 14 (clear DTC command)
 FF,00 : all group clear
 CS : checksum of incoming FORD9141 message

Commande	12
Fonction	Get express sensor data from Ford9141 BUS
Remarque	This message is sent to the Modul pointed by command 10 User needs only to send the 2 bytes of PID No header sending is necessary. MOByDic uses already existing header.
Request	12 , PID_hb , PID_lb (hb=high byte , lb=low_byte)
Pos. Response	12 , length , Header1 , 0xF1 , Ecu_address , <data> , CS
Neg. Response	N/A

Exemple for command 12 how to get a sensor value using the PID from pointed ECU

REQUEST LATERAL ACCELEROMETRE SENSOR FROM ABS MODUL USING COMMAND 12

```
write_rs232(12); // send MESSAGE command
write_rs232($3A); // 3 Bytes follow
write_rs232($51); // PID = 3a51 for this sensor.

rs232_buffer[0]:=read_rs232(250); // wait for 1.byte
if read_error then exit // no response
if rs232_buffer[0] <> 12 the exit // false sync.
```



```

Length := read_rs232(10);           // get incoming message length
For i:=1 to length do
  Begin
    rs232_buffer[i]:=read_rs232(10); // read length x byte 10 ms interbyte timeout
  End;

```

In Rs232_buffer user has following data :

12 , 09 , 54 , F1 , 28 , 62 , pp , pp , dd , dd , CS

- 12 : response to command 12 (echo)
- 09 : total bytes follow
- 84 : type of ford9141 message header
- F1 : targed address (mobydic)
- 28 : source address (ABS)
- 62 : response to read by PID parameter
- pp,pp : requested PID confirmation
- dd,dd : sensor value
- CS : checksum of incoming FORD9141 message

If a 7F is sent instead of dd dd , the PID with whis number is not implemented.

Commande	13
Fonction	EOBD handling on ford9141
Remarque	This command is reserved for future use
Request	
Pos. Response	
Neg. Response	

Commande	14 Connect to ISO9141 ECU
Fonction	Connect to ISO ECU with existing Header
Remarque	This command is reserved for future use
Request	
Pos. Response	
Neg. Response	



Commande	20
Fonction	Set ECU Address on FORD SCP BUS
Remarque	Default setting is \$10 (PCM = powertrain control modul)
Request	20 , ECU Address
Pos. Response	ACK (06)
Neg. Response	N/A

Exemple for command 20 how setting the header in mObydic to PCM on SCP Bus

```

Open_COMM;
clearbuffers;
write_rs232(20); // send set address command
write_rs232($10); // send PCM address to mOByDic
rs232_buffer[0]:=read_rs232(100); // get the response with 100 ms timeout
if read_error then exit // no response
if rs232_buffer[0] = ACK
    then ..... // handle
    
```

Exemple for command 20 how setting the header of Cluster Panel Modul on SCP Bus

```

Open_COMM;
clearbuffers;
write_rs232(20); // send set address command
write_rs232($60); // send cluster panel address to mOByDic
rs232_buffer[0]:=read_rs232(100); // get the response with 100 ms timeout
if read_error then exit // no response
if rs232_buffer[0] = ACK
    then ..... // handle
    
```



Commande	21
Fonction	Send a free message to Ford SCP BUS
Remarque	This message is sent to the Modul pointed by command 20 User needs only to send the bytes in data field of ford message format. No header sending is necessary. MOByDic uses already existing header.
Request	21 , data_length , <data>
Pos. Response	21 , length , Header1 , 0xF1 , Ecu_address , <data> , CS
Neg. Response	Data_length byte doesnt match the data length

Exemple for command 21 get the number of DTC from by command 20 pointed modul

FIRST POINT TO PCM MODUL

```

Open_COMM;
clearbuffers;
write_rs232(20); // send set address command
write_rs232($10); // send PCM address to mOByDic
rs232_buffer[0]:=read_rs232(100); // get the response with 100 ms timeout
if read_error then exit // no response
if rs232_buffer[0] <> ACK then exit;

```

REQUEST GET NUMBER OF DTC FROM PCM MODUL USING COMMAND 21

```

write_rs232(21); // send MESSAGE command
write_rs232(3); // 3 Bytes follow
write_rs232($22); // 22 is get value by PID
write_rs232($02); //
write_rs232($00); // 0200 is the parameter for nmb. Of dtc.

rs232_buffer[0]:=read_rs232(250); // wait for 1.byte
if read_error then exit // no response

```



```

if rs232_buffer[0] <> 21 the exit // false sync.
Length := read_rs232(10); // get incoming message length
For i:=1 to length do
  Begin
    rs232_buffer[i]:=read_rs232(10); // read length x byte 10 ms interbyte timeout
  End;

```

In Rs232_buffer user has following data :

21 , 08 , C4 , F1 , 10 , 62 , 02 , 00 , xx , CRC

- 21 : response to command 21 (echo)
- 08 : total bytes follow
- C4 : type of SCP message header
- F1 : targeted address (mobydic)
- 10 : Source address (PCM)
- 62 : response to 22 get value by PID
- 02,00 : requested PID confirmation
- xx : nmb of DTC
- CRC : SCP CRC summ

Commande	22
Fonction	Get express sensor data from SCP BUS
Remarque	This message is sent to the Modul pointed by command 20 User needs only to send the 2 bytes of PID No header sending is necessary. MOByDic uses already existing header.
Request	22 , PID_hb , PID_lb (hb=high byte , lb=low_byte)
Pos. Response	22 , length , Header1 , 0xF1 , Ecu_address , <data> , CS
Neg. Response	N/A

Exemple for command 22 how to get a sensor value using the PID from pointed ECU

REQUEST BATTERY VOLTAGE FROM PCM MODUL USING COMMAND 22

```

write_rs232(22); // send MESSAGE command
write_rs232($11); //
write_rs232($72); // PID = 0x1172 for this sensor.

rs232_buffer[0]:=read_rs232(250); // wait for 1.byte
if read_error then exit // no response
if rs232_buffer[0] <> 22 the exit // false sync.
Length := read_rs232(10); // get incoming message length
For i:=1 to length do
  Begin
    rs232_buffer[i]:=read_rs232(10); // read length x byte 10 ms interbyte timeout
  End;

```

In Rs232_buffer user has following data :

22 , 09 , C4 , F1 , 10 , 62 , pp , pp , dd , dd , CRC

- 22 : response to command 22 (echo)
- 09 : total bytes follow
- C4 : type of SCP message header
- F1 : targeted address (mobydic)
- 10 : Source address (PCM)
- 62 : response to 22 get value by PID
- 11,72 : requested PID confirmation



dd,dd : PID value
CRC : SCP CRC summ

If a 7F is sent instead of dd dd , the PID with whis number is not implemented.

Commande	23
Fonction	Set Header and incoming message filter
Remarque	This message allows access to any OBD compliant ECU
Request	23 , HDR1,HDR2,HDR3,IMF1,IMF2,IMF3
Pos. Response	ACK
Neg. Response	N/A

Exemple for command 23 how To set an OBDII/EOBD Header / filter

```
write_rs232(23);           // send MESSAGE command
write_rs232($61);         // header 1
write_rs232($6a);         // header 2
write_rs232($F1);         // header 3
write_rs232($41);         // incoming message header 1 filter
write_rs232($6b);         // incoming message header 2 filter
write_rs232($10);         // incoming message header 3 ( only PCM )
```

```
rs232_buffer[0]:=read_rs232(250); // wait for 1.byte
if read_error then exit           // no response
if rs232_buffer[0] <> ACK the exit // false sync.
```

Exemple for command 21 how to communicate in EOBD Mode (service 1 PID 0)

```
write_rs232(21);           // send MESSAGE command
write_rs232(2);            // 2 Bytes follow
write_rs232($01);          // Service 0
write_rs232($00);          // PID 0
```

```
rs232_buffer[0]:=read_rs232(250); // wait for 1.byte
if read_error then exit           // no response
if rs232_buffer[0] <> 21 the exit // false sync.
Length := read_rs232(10);         // get incoming message length
For i:=1 to length do
  Begin
    rs232_buffer[i]:=read_rs232(10); // read length x byte 10 ms interbyte timeout
  End;
```

In Rs232_buffer user has following data :

21 , 0a , 41 , 6b , 10 , 41 , 00 , xx , xx , xx , xx , CRC



```

21 : response to command 21 ( echo )
0a : total bytes follow
41 : Functional addressing response
6b : Functional addressing response
10 : Source address ( PCM )
41 : Response to service 01
00 : confirmation PID 0
10 : Source address ( PCM )
xx : 4 byte response
CRC : SCP CRC summ
    
```

Commande	30
Fonction	Set ECU Address on FORD CAN BUS
Remarque	Default setting is \$7E0 (PCM = powertrain control modul)
Request	30 , ECU_Address
Pos. Response	ACK (06)
Neg. Response	N/A

When using this command , an incoming filter is automatically set for response header. if the user sets the header to 07E0 (PCM) the incoming filter is set to 7E8 by mobydic. Any access to EOBD moduls with the header 7DF results the setting of the incoming filter to 7E8...7EF automatically

Exemple for command 30 how setting the header in mObydic to TCM on CAN Bus

```

Open_COMM;
clearbuffers;
write_rs232(30); // send set address command
write_rs232($07); // send TCM address to mOByDic
write_rs232($E1); // send TCM address to mOByDic
rs232_buffer[0]:=read_rs232(100); // get the response with 100 ms timeout
if read_error then exit // no response
if rs232_buffer[0] = ACK
    then ..... // handle
    
```

Exemple for command 30 how setting the header in mObydic to ABS on CAN Bus

```

Open_COMM;
clearbuffers;
write_rs232(30); // send set address command
write_rs232($07); // send ABS address to mOByDic
write_rs232($60); // send ABS address to mOByDic
rs232_buffer[0]:=read_rs232(100); // get the response with 100 ms timeout
if read_error then exit // no response
if rs232_buffer[0] = ACK
    then ..... // handle
    
```



Commande	31
Fonction	Send a free message to Ford CAN BUS
Remarque	This message is sent to the Modul pointed by command 30 User needs only to send the bytes in data field of ford message format. No header sending is necessary. MOByDic uses already existing header. PCI handling is made by mOByDic.
Request	31 , data_length , <data>
Pos. Response	31 , length , Header , <data>
Neg. Response	Data_length byte doesnt match the data length

Exemple for command 31 how to get a PID Value from, by command 30 pointed modul (Navigation display)

FIRST POINT TO Navigation MODUL

```
Open_COMM;
clearbuffers;
write_rs232(30); // send set address command
write_rs232($07); // send NAVI address to mOByDic
write_rs232($73); // send NAVI address to mOByDic
rs232_buffer[0]:=read_rs232(100); // get the response with 100 ms timeout
if read_error then exit // no response
if rs232_buffer[0] <> ACK then exit;
```

Get Volume control rotary position

```
write_rs232(31); // send MESSAGE command
write_rs232(3); // 3 Bytes follow
write_rs232($22); // 10 is entry
write_rs232($81); //
write_rs232($33); // 8133 is the PID

rs232_buffer[0]:=read_rs232(250); // wait for 1.byte
if read_error then exit // no response
if rs232_buffer[0] <> 31 the exit // false sync.
Length := read_rs232(10); // get incoming message length
For i:=1 to length do
  Begin
    rs232_buffer[i]:=read_rs232(10); // read length x byte 10 ms interbyte timeout
  End;
```

In Rs232_buffer user has following data :

31 , 07 , 07 , 7B , 04 , 62 , 81 , 33 , xx

31 : response to command 31 (echo)



07 : total bytes follow
 07,7B : CAN response header
 04 : PCI
 62 : response to 22 get value by PID
 81,33 : requested PID confirmation
 xx : value

Commande	32
Fonction	Get express sensor data from CAN BUS
Remarque	This message is sent to the Modul pointed by command 30 User needs only to send the 2 bytes of PID No header sending is necessary. MOByDic uses already existing header. PCI handling is made by mOByDic.
Request	32 , PID_hb , PID_lb (hb=high byte , lb=low_byte)
Pos. Response	32 , length , Header , <data>
Neg. Response	N/A

Exemple for command 32 how to get a sensor value using the PID from pointed ECU

REQUEST BAROMETER SENSOR FROM PCM MODUL USING COMMAND 32

POINT TO PCM

```
Open_COMM;
clearbuffers;
write_rs232(30); // send set address command
write_rs232($07); // send PCM address to mOByDic
write_rs232($E0); // send PCM address to mOByDic
rs232_buffer[0]:=read_rs232(100); // get the response with 100 ms timeout
if read_error then exit // no response
if rs232_buffer[0] <> ACK then exit;
```

Get baro sensor

```
write_rs232(31); // send MESSAGE command
write_rs232($11); //
write_rs232($42); // 1142 is the PID

rs232_buffer[0]:=read_rs232(250); // wait for 1.byte
if read_error then exit // no response
if rs232_buffer[0] <> 31 the exit // false sync.
Length := read_rs232(10); // get incoming message length
For i:=1 to length do
  Begin
    rs232_buffer[i]:=read_rs232(10); // read length x byte 10 ms interbyte timeout
  End;
```

In Rs232_buffer user has following data :

31 , 0a , 07 , E8 , 05 , 62 , 11 , 42 , xx , xx , 00 , 00

31 : response to command 31 (echo)
 0a : total bytes follow
 07,E8 : CAN response header
 05 : PCI
 62 : response to 22 get value by PID
 11,42 : requested PID confirmation
 xx,xx : value



Exemple for EOBD Access

SET THE EOBD HEADER

```
Open_COMM;
clearbuffers;
write_rs232(30); // send set address command
write_rs232($07); // send EOBD Header to mOByDic
write_rs232($df); //
rs232_buffer[0]:=read_rs232(100); // get the response with 100 ms timeout
if read_error then exit // no response
if rs232_buffer[0] <> ACK then exit;
```

Get Service 1 PID 0 Message

```
write_rs232(31); // send MESSAGE command
write_rs232(2); // 2 Bytes follow
write_rs232($01); // Service 1 PID 0 request
write_rs232($00); //

rs232_buffer[0]:=read_rs232(250); // wait for 1.byte
if read_error then exit // no response
if rs232_buffer[0] <> 31 the exit // false sync.
Length := read_rs232(10); // get incoming message length
For i:=1 to length do
  Begin
    rs232_buffer[i]:=read_rs232(10); // read length x byte 10 ms interbyte timeout
  End;
```

In Rs232_buffer user has following data :

31 , 0a , 07 , E8 , 06 , 41 , 00 , xx , xx , xx , xx , 00

```
31 : response to command 31 ( echo )
0a : total bytes follow
07,E8 : CAN response header
06 : PCI
41 : Response to service 1
00 : requested PID confirmation
xx : implemented PIDs
```

MOByDic has an internal CAN Buffer of 256 bytes. Handling till 32 multiframe is also possible.

Exemple for Multiframe response

SET THE PCM HEADER



```

Open_COMM;
clearbuffers;
write_rs232(30);           // send set address command
write_rs232($07);        // send PCM Header to mOByDic
write_rs232($E0);        //
rs232_buffer[0]:=read_rs232(100); // get the response with 100 ms timeout
if read_error then exit // no response
if rs232_buffer[0] <> ACK then exit;

```

Get DTCs

```

write_rs232(31);           // send MESSAGE command
write_rs232(3);           // 3 Bytes follow
write_rs232($13);        // Get DTCs
write_rs232($ff);        //
write_rs232($00);        //

rs232_buffer[0]:=read_rs232(250); // wait for 1.byte
if read_error then exit // no response
if rs232_buffer[0] <> 31 the exit // false sync.
Length := read_rs232(10); // get incoming message length
For i:=1 to length do
  Begin
    rs232_buffer[i]:=read_rs232(10); // read length x byte 10 ms interbyte timeout
  End;

```

In Rs232_buffer user has following data :

```

31 , 12 , 07 , E8 , 07 , 53 , nmb , dtc1 ,dtc1 , dtc2 , dtc2 , 00 , sequence , dtc3
, dtc3 , 00 , 00 , 00 , 00 , 00

```

```

31 : response to command 31 ( echo )
12 : total bytes follow decimal=18
07,E8 : CAN response header
07 : PCI
53 : Response to request mode
nmb : nombre of DTC
DTCs : fault codes
Sq : mark of next message

```

Commande	33
Fonction	Adjust IDTAG / incoming CAN filter
Remarque	Default setting is
Request	33 , IDTagHB , IDTagLB
Pos. Response	ACK (06)
Neg. Response	N/A



Exemple for command 33 how setting the IDTag and IDMask

```
Open_COMM;
clearbuffers;
write_rs232(33); // send set address command
write_rs232($07); // set filter on incoming message
write_rs232($E8); // accept only data from 7e8
rs232_buffer[0]:=read_rs232(100); // get the response with 100 ms timeout
if read_error then exit // no response
if rs232_buffer[0] = ACK
  then ..... // handle
```

Commande	40
Fonction	FORmiDable Handling
Remarque	Reserved for FORmiDable
Request	
Pos. Response	
Neg. Response	

