FMS OEM CHIP V5+

Content:

Ing. David Španěl

Mgr. Vítězslav Rejda

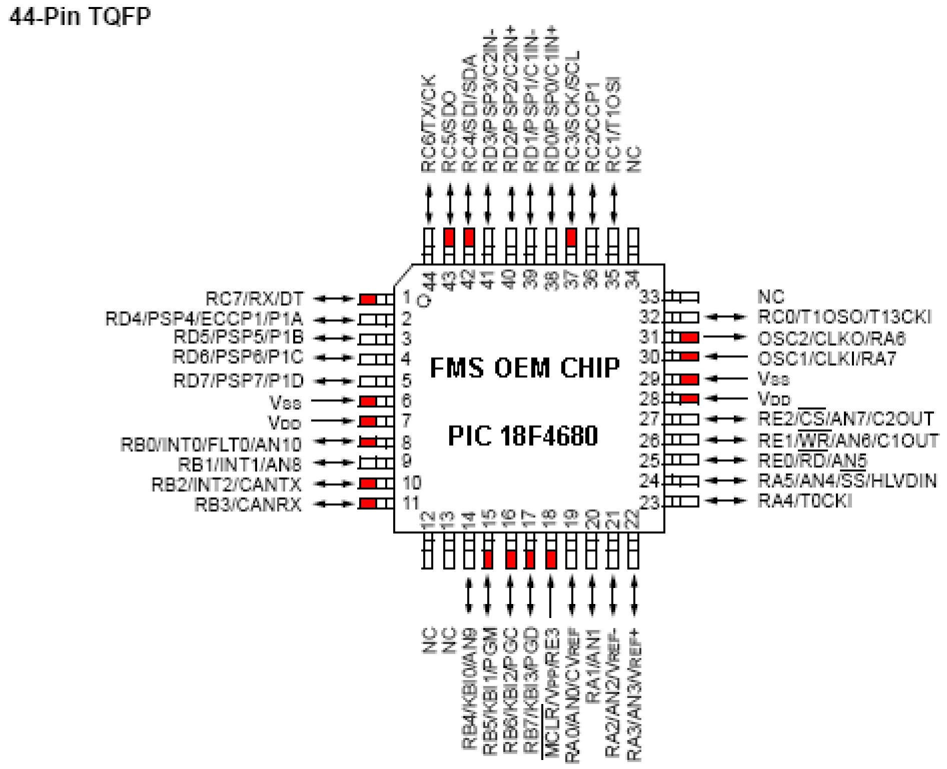
CANLAB s.r.o.

General description

FMS OEM CHIP is preset microcontroller PIC 18F4680 in 44-pin TQFP case designed for integration into the system for car monitoring, systems for car fleet administration and so on. The chip performs pre-processing of data from the car CAN bus and, if applicable, from DTCO 1381 digital tachograph to which it is connected by means of its info interface. Current firmware version supports data processing from CAN bus of cars that use SAE J1939 protocol and data processing from CAN bus of some VW cars (including Škoda), Mercedes Sprinter, Nissan and so on. It supplies these pre-processed data into superordinate system by means of SPI bus functioning in SPI slave mode, or by means of RS232 interface (UART).

Version 5 of this chip allows reading of car error codes and contains support for direct user data reading from CAN or sending data to CAN.

On demand, different car equipped with CAN bus can be imported into the firmware. It is necessary to have the car available for a while so the measurement for the identification of data required at the CAN bus of this car can be done.



Picture 1: FMS OEM CHIP case

Integration of FMS chip into the system

To work correctly, the circuit requires supply potential of 5 V and 10 MHz clock frequency supply. It is recommended to provide circuit programming connector to do circuit firmware update.

During design stage, it is recommended to take optional switch on/switch off of CAN bus terminal resistance of 120 ohm into account. This terminal resistance is used when connecting to so called FMS gate of trucks. CAN bus is usually terminated on each side by terminal resistance of 120 ohm (between CAN H and CAN L is measured resistance of 60 Ohm).

It is recommended to connect MCLR signal as external reset of circuit from superordinate output. When CS is not active, SDO signal is switched as input.

Two LED indicators can be connected to the chip. LED connected to RC0 pin indicates CAN activity. During message receiving that includes data about car speed the output changes its status. LED connected to RC1 pin indicates tachograph activity. Output changes the status during receiving of valid message from tachograph.

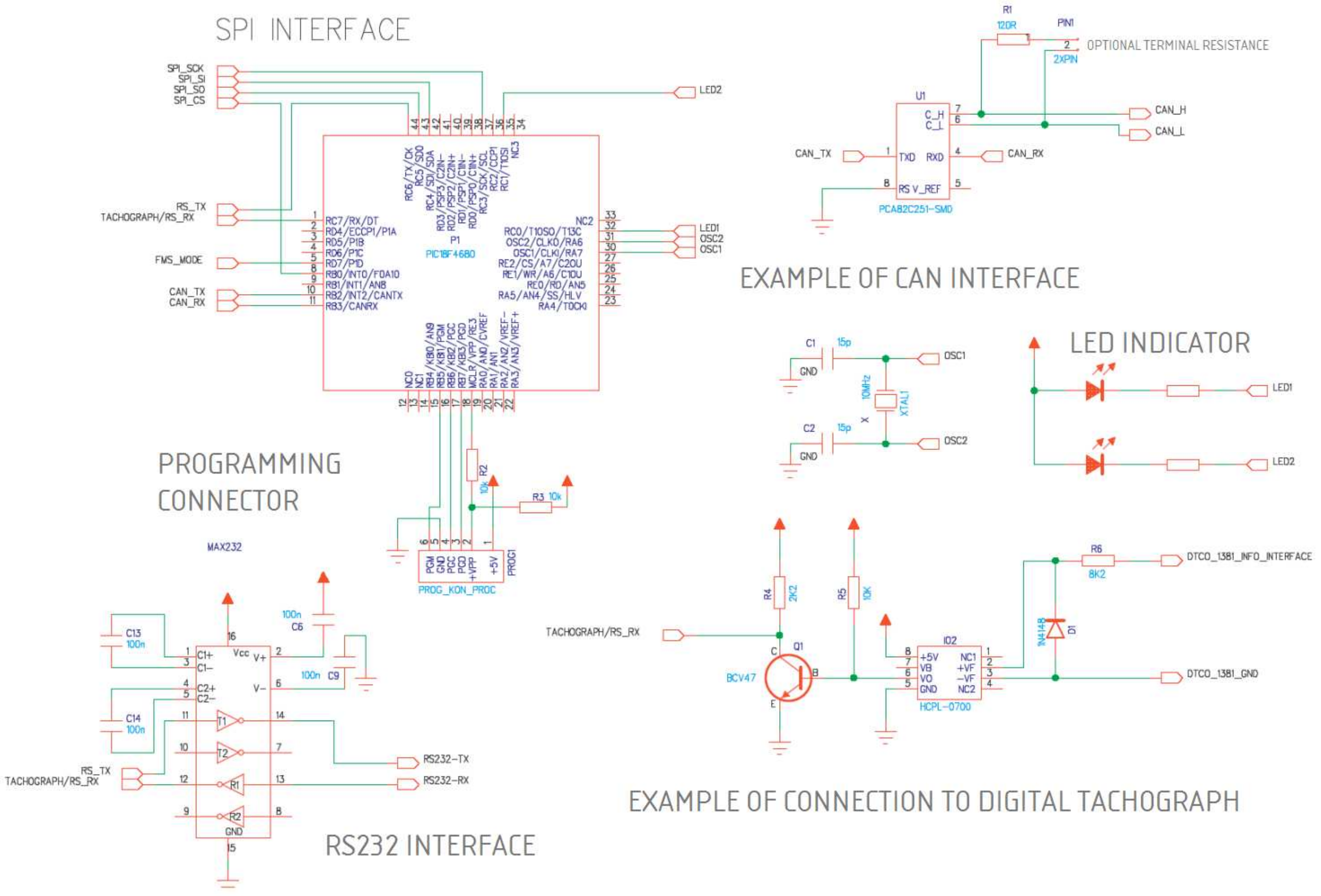
After switching on, CAN interface is active or inactive; if you require active interface without necessity of its initialization, you must specify this function in your order – then the setting is saved in internal EEPROM. **It is necessary to properly set the device before installation into the car.** Tachograph input is always active after switching on and processes data immediately after connecting voltage and chip internal initialization (up to 100 ms according to the firmware variant).

Chip provides data from CAN bus that are accessible via tapping. Different message sets are available for various cars, models and years of manufacture.

Interface to superordinate system

For connecting to superordinate system (that uses FMS OEM CHIP for acquiring data from the car) you can use one of two possible interfaces: SPI or RS232. Used interface selection is made by means of logical level that is induced to RD7 pin. After switching on, if there is logical zero (0), RS232 interface is used. In case of logical 1 SPI interface is used by the superordinate system.

RS232 mode allows connection of CAN bus only. SPI mode allows connection of CAN bus and digital tachograph.

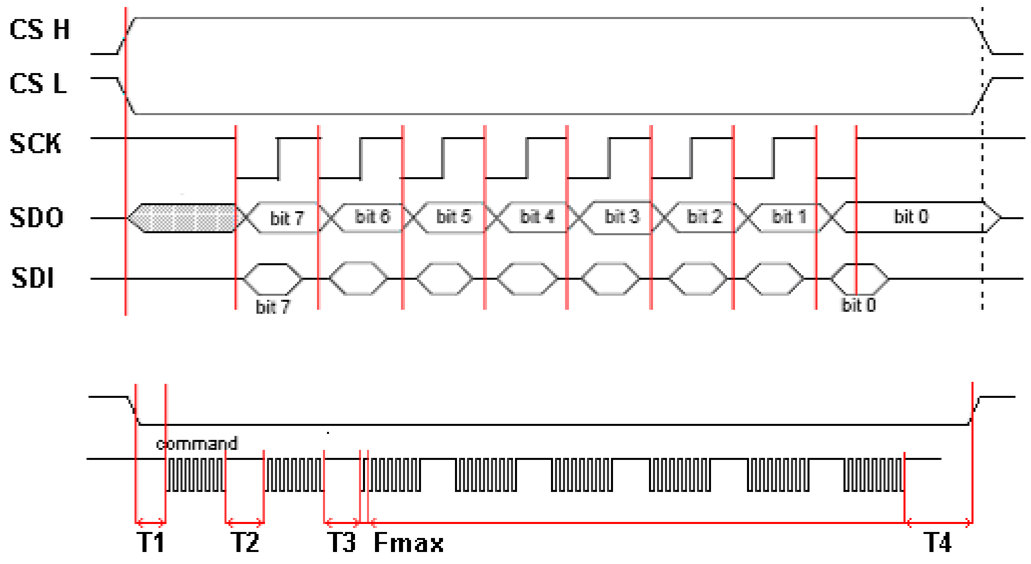


Picture 2: Typical circuit connection.

SPI interface

FMS OEM CHIP provides data via SPI interface. Interface works in SPI mode 01. It is possible to supply the circuit variant with active level of CS signal in H status. Maximum SPI speed (F_{SCKMAX}) is 2 MHz. During communication through SPI, it is necessary to observe these rules:

- F_{SCK} <= 3.00 MHz
- Time between CS signal transition into active level (logical 0) and sending/reading of the first byte (T1) must be >= 12 uS.
- Time between termination of command byte transmitting and commencement of reading of the first transfer data byte (T2) >= 12 uS.
- Time between termination of transfer data byte and reading of next data byte (T3) >= 1 uS.
- Time after transfer of the last data byte and CS deactivation (T4) >= 10 uS.
- To ensure speed of data entry into the SPI internal registry in SPI slave mode, data processing from CAN bus is stopped at the moment of active CS signal. Data receiving to HW registers of integrated CAN controller is active, thus, there is no significant data loss in reality. CAN bus periodically repeat the important data, so it will be actualized later at worst. Its length depends on length of CS signal activation.
- It is recommended to insert interval of 1 ms among two subsequent readings (CS deactivation) during the first reading and CS activation.



CS H – Firmware type with active signal CHIP SELECT, logical 1.
CS L – Firmware type with active signal CHIP SELECT, logical 0.
Customer chooses CS signal type according to his needs.

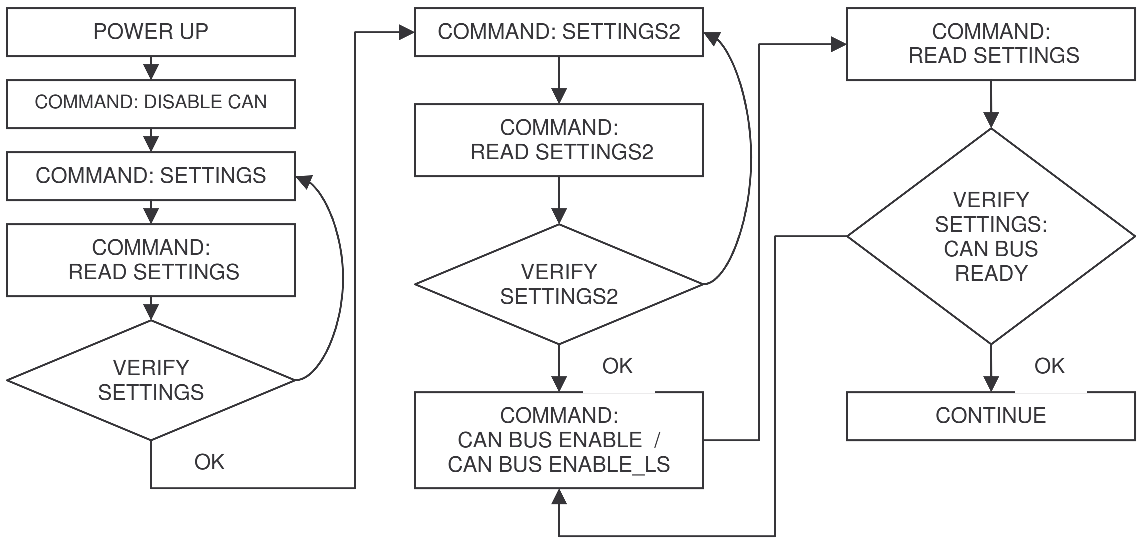
Picture 3: SPI interface

Master starts the transfer to SPI by sending “command” byte. This byte specifies next action. This byte can be followed by scope of data read from FMS OEM CHIP. Respective commands are specified in following sheet:

Command	**Command byte value (binary)** **b7 b0**	**Description**
READ	0000 0XXX Some of the below stated data structures follows.	XXX: 000 – Firmware version 001 – CAN bus data 010 – Tachograph data 110 – Settings 2 111 – Settings
SETTINGS	01ZY YXXX 0011 0011 (51d) 1010 0101 (165d)	Z: 0 – Normal CAN mode 1 – Listen only CAN mode YY: 00 – Standard 11bit CAN ID 01 – Expanded 29bit CAN ID 11 – Standard and expanded 29bit CAN ID XXX: CAN bus speed 0001 – 62.5k 0010 – 83.3k 0011 – 100k 0100 – 125k 0101 – 250k 0110 – 500k 0111 – 1M
SETTINGS2	1XXX XXXX (low byte) XXXX XXXX (high byte) RRRR RYYY 1010 0101 (165d) 0011 0011 (51d)	X-additional information for choosing car type (CAR TYPE). This information is required in case of the identical information in CAN message with the same identifier for two different cars. Y-type tachograph,0-VDO Siemens, 1-Stoneridge,2-Actia R-reserve
CAN BUS ENABLE	0001 0001 (17d) 0011 1100 (60d) 0101 1010 (90d)	Enables CAN bus. Listen only according to settings.
CAN BUS ENABLE LS	0001 0010 (18d) 0111 1110 (126d)	Forces enabling of CAN bus only in listen-only mode.

	1010 0101 (165d)	
CAN BUS DISABLE	0001 0000 (16d) 1010 0101 (165d) 0000 1111 (15d)	Disables CAN bus.
SEND OBD DTC REQ.	0001 1000 (24d) 0000 0XXX	Sending of OBD request. OBD DTC request code. 1- 11bit ID, mode 3 2- 11bit ID, mode 7 3- 29bit ID, mode 3 4- 29bit ID, mode 7
READ OBD DTC TABLE	0001 1111 (31d) Data structure described below follows.	Reading of OBD DTC codes in the memory.
SET CAN USR RCV MSG	0001 0011 (19d) Data structure described below follows.	Command for user buffer settings for receiving messages from CAN.
DISABLE ALL CAN USR RCV MSG	0001 0100 (20d)	Command disables all user buffers for receiving messages from CAN.
SEND CAN USR MSG	0001 0110 (22d) Data structure described below follows.	Command for user sending data to CAN.
READ CAN USR RCV MSG	0001 0111 (23) 0000 0AAA Data structure described below follows.	Command for user reading data from CAN – reading of user buffer. AAA-buffer index 1..7

SETTINGS Command at SPI



READ Command at SPI

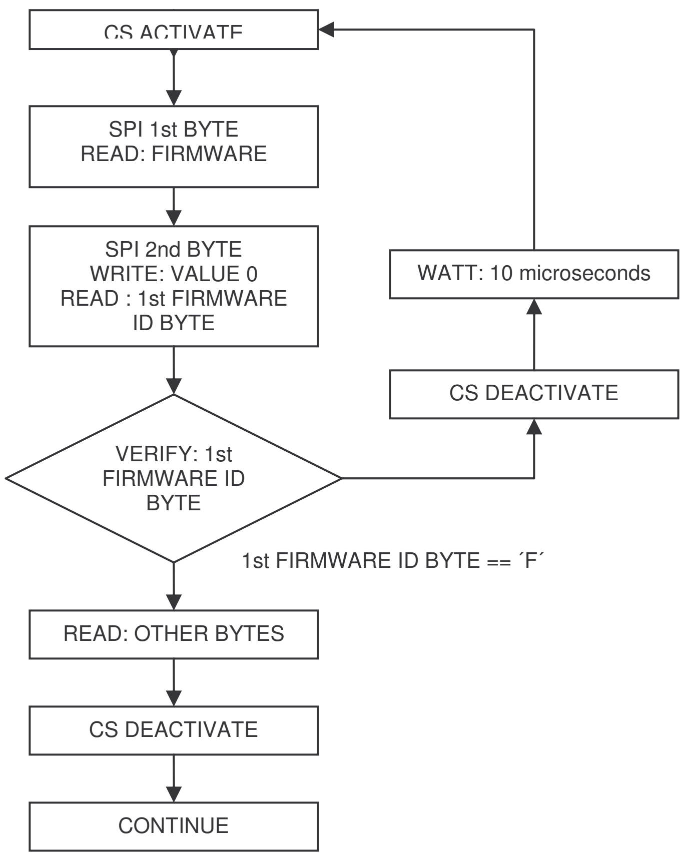
Command READ allows block data reading from FMS OEM CHIP. Master sends the first data byte with command READ and receive value 0. Next reading of SPI allows receiving of data. Reading is finished after reading of the whole data structure, or it can be terminated sooner through returning the CS signal do inactive level (logical 1).

READ: FIRMWARE

This command allows reading of the FMS OEM CHIP firmware version. Length of read data: 13 bytes. Thus, the total length of the transfer is 14 bytes, 1 byte for READ FIRMWARE command and 13 data bytes. Returned value corresponds to text chain with the firmware version, e.g. “FMSOEMV5002CT__”. This chain is not finished with finishing sign.

FMSOEMV5002CT__ (FMSOEMV5yyyab__)

yyy	- firmware version
a	- C-supports CAN, X-do not support CAN
b	- T-supports digital tachographs, X-does not support digital tachographs
__	- reserve (2 digits)



READ: CAN BUS DATA

This command allows data reading from CAN interface. Data forming following structure are read:

```
#define BYTE      unsigned char     // 8bit unsigned data type
#define UINT      unsigned __int16  // 16bit unsigned data type
#define SINT      signed   __int16  // 16bit signed data type
#define ULONG     unsigned __int32  // 32bit unsigned data type
```

Multi-byte types are in little-endian format.

FMS structure definition up till version 1.10 (incl.):

```
typedef struct _FMS {
            BYTE  begin_check;            // size of(FMS) = 0x52
            UINT  rpm;
            UINT  speed;
            BYTE  acc_pedal;
            BYTE  brake_pedal;
            ULONG total_fuel_used;
            ULONG total_engine_hours;
            BYTE  fuel_level;
            UINT  fuel_consumption;
            BYTE  axle_weight_captured[12];
            BYTE  axle_weight_location[12];
            UINT  axle_weight[12];
            ULONG total_vehicle_distance;
            UINT  daily_vehicle_distance;
            SINT  service_distance;
            BYTE  engine_coolant_temperature;
            UINT  tachograph_speed;
            BYTE  tachograph[4];
            BYTE  end_check;              // 0xAA
} FMS;
```

FMS structure definition from version 1.11 (incl.):

```
typedef struct _FMS {
            BYTE  begin_check;            // size of(FMS) = 0x7C
            UINT  rpm;
            UINT  speed;
            BYTE  acc_pedal;
            BYTE  brake_pedal;
            ULONG total_fuel_used;
            ULONG total_engine_hours;
            BYTE  fuel_level;
            UINT  fuel_consumption;
            BYTE  axle_weight_captured[12];
            BYTE  axle_weight_location[12];
            UINT  axle_weight[12];
            ULONG total_vehicle_distance;
            UINT  daily_vehicle_distance;
            SINT  service_distance;
            BYTE  engine_coolant_temperature;
            UINT  tachograph_speed;
            BYTE  tachograph[4];
            BYTE  tire_pressure_captured[12];
            BYTE  tire_pressure_location[12];
            BYTE  tire_pressure[12];
            UINT  door;
            UINT  fuel_instantaneous;
            UINT  fuel_rate;
            BYTE  end_check;

} FMS;
```

Valid value of begin_check and end_check indicates valid data. If these two items do not have correct values, the car data must be ignored. If all item bytes are set at 0xFF, non-acceptance of these data from Can is indicated.

It is possible to convert the data from the structure into real values by means of this sheet:

Data	**Number of bits**	**Value of 1 bit**	**Offset**
Speed	16	1/256 km/h	0
Acceleration pedal position	8	0.4 %	0
Brake pedal position	8	0.4 %	0
Total fuel consumption	32	0.5 liter	0
Fuel tank level	8	0.4 %	0
Engine rpm	16	0.125 round	0
Axle weight	16	0.5 kg	0
Total number of moto-hours	32	0.05 h	0
Total number of kilometers	32	5 m	0
Distance (km) to the next service control	16	5 km	-160 635
Cooling liquid temperature	8	1 °C	-40

Mean consumption	16	1/512 km/L – SAE1939 L/100km - VW	0

Axle weight

Item axle_weight_location[x] states axle weight value location of item axle_weight[x]. Value of axle_weight_captured[x] = 0 means that this item do not contain any (valid) value, axle_weight_captured[x] = 1 means that this item contains valid value.

In the item axle_weight_location[x], information about number of measured axle and its wheel is coded. Lower 4 bits state wheel index, upper 4 bits state axle index. If all 4 bits are set at 1, location is not known.

Tachograph[4]

This item contains information that can be decoded following this steps:

tachograph[0]
Bit 2..0: Driver 1 working state
000 = Rest
001 = Driver available
010 = Work
011 = Drive
110 = Error
111 =
Bit 5..3: Driver 2 working state
000 = Rest
001 = Driver available
010 = Work
011 = Drive
110 = Error
111 = not available
Bit 7..6: Drive recognize
00 = Vehicle motion not detected
01 = vehicle motion

tachograph[1]
Bit 3..0: Driver 1 time rel states
0000 = normal
0001 = 15 min bef. 4 ½ h
0010 = 4 ½ h reached
0011 = 15 min bef. 9 h
0100 = 9 h reached
0101 = 15 min bef. 16 h
0110 = 16h reached
1110 = Error
1111 = not available
Bit 5..4: Driver 1 card

00 = Card not present
01= Card present
Bit 7..6: Overspeed
00 = No overspeed
01 = Overspeed

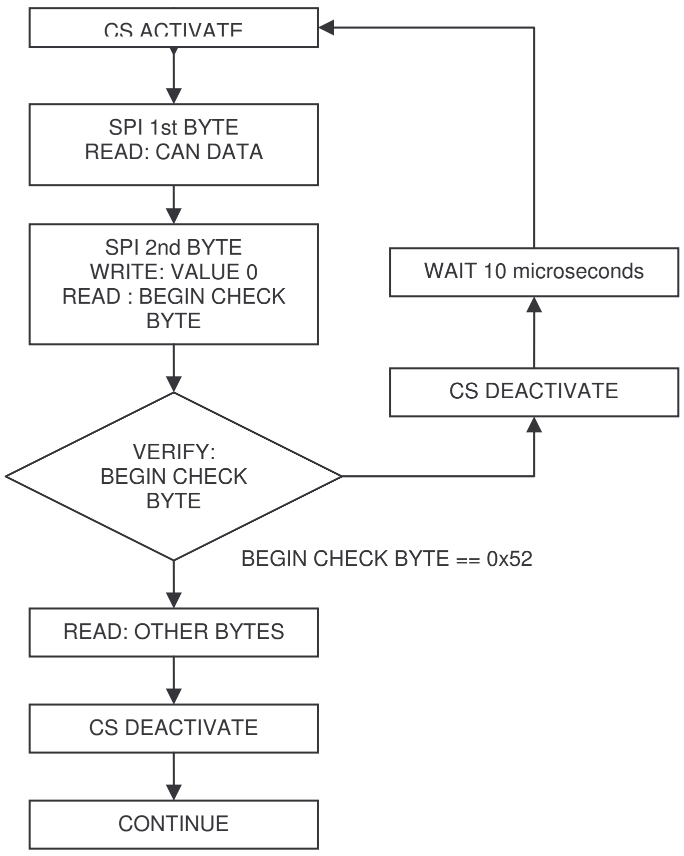
tachograph[2]
Bit 3..0: Driver 2 time rel states
0000 = normal
0001 = 15 min bef. 4 ½ h
0010 = 4 ½ h reached
0011 = 15 min bef. 9 h
0100 = 9 h reached
0101 = 15 min bef. 16 h
0110 = 16h reached
1110 = Error
1111 = not available
Bit 5..4: Driver 2 card
00 = Card not present
01= Card present
Bit 7..6: Not used

tachograph[3]
Bit 0..1: System event
00 = no tachogr. Event
01 = tachogr. Event

Bit 2..3: Handling information
00 = no handling information
01 = handling information

Bit 5..4: Tachograph performance
00 = Normal performance
01 = Performance

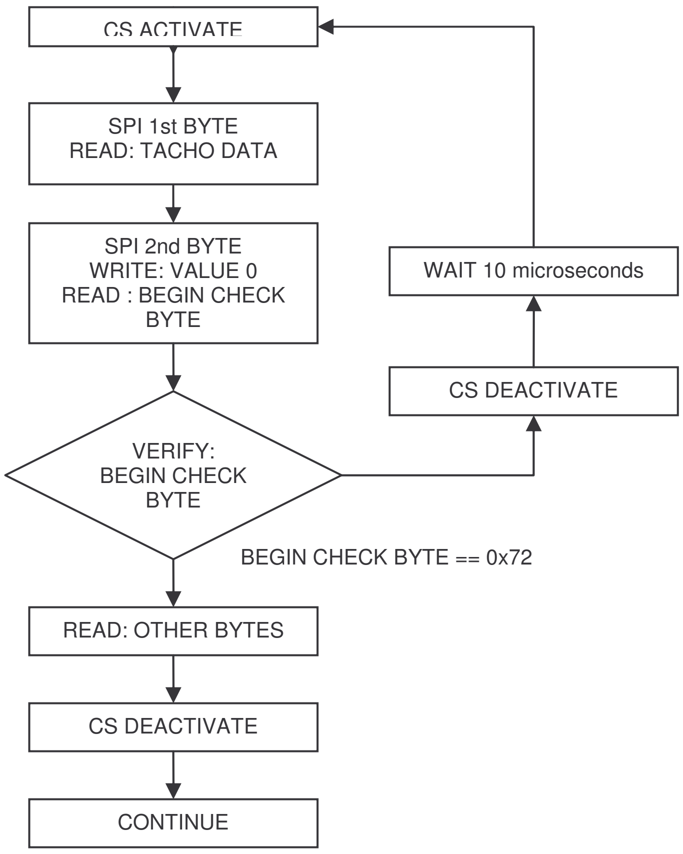
Bit 7..6: Direction indicator
00 = Forward
01 = Reverse



READ: TACHOGRAPH DATA

```
typedef struct _DTI{
            BYTE        begin_check;            // size of(DTI) = 0x72
            BYTE        seconds;
            BYTE        minutes;
            BYTE        hours;
            BYTE        month;
            BYTE        day;
            BYTE        year;
            BYTE        local_minute_offset;
            BYTE        local_hour_offset;
            BYTE        work_states;
            BYTE        driver_1_states;
            BYTE        driver_2_states;
            BYTE        tachograph_status;
            UINT        tachograph_vehicle_speed;
            LONG        total_vehicle_distance;
            LONG        trip_distance;
            INT         k_factor;
            INT         engine_speed;
            INT         additional_information;
            BYTE        vehicle_id_len;
            BYTE        vehicle_id[20];
            BYTE        vehicle_reg_len;
            BYTE        vehicle_reg[20];
            BYTE        driver_1_len;
            BYTE        driver_1[20];
            BYTE        driver_2_len;
            BYTE        driver_2[20];
            BYTE        end_check;              // 0xAA
```

```
    } DTI;
```



READ: SETTINGS

This command is used for regressive reading of actual settings. Answer is one data byte with following bit meaning:

AAZY YXXX

AA:	00 – CAN bus OFF
	01 – CAN bus ON
	10 – Startup CAN
Z:	0 – Normal CAN mode
	1 – Listen only CAN mode
YY:	00 – Standard 11bit CAN ID
	01 – Expanded 29bit CAN ID
	10 – Standard and expanded 29 bit CAN ID
XXX:	CAN bus speed
	0001 – 62.5k
	0010 – 83.3k
	0011 – 100k
	0100 – 125k
	0101 – 250k

0110 – 500k
0111 – 1M

READ: SETTINGS2

After the first data byte with reading command SETTINGS2, two data bytes with car type and next byte with tachograph type follow.

Car	Car type	CAN
FMS/SAE1939	0	250k, ext
VW	0	500k, st
OBD 11bit	1024	250/500k, st
OBD 29bit	1025	250/500k, ext
Mercedes Sprinter	48	500k, st
Mercedes Vito	49	500k, st
Nisan/Renault	64	250/500k, st
Nisan/Renault	65	250/500k, st
Toyota	96	500k, st
Ford Mondeo	128	500k, st
Ford Tranzit	129	500k, st
Ford CMAX	130	500k, st
Fiat	144	250/500k, st
Mazda	160	500k, st
Suzuki SX4	176	500k, st

READ OBD DTC TABLE

```
typedef struct _OBD_DTC_TABLE{
            BYTE        size;
            INT         dtc[16];
} OBD_DTC_TABLE;
```

Below mentioned structure follows after command for error code sheet reading. The first byte contains number of valid DTC codes in the sheet. Maximum 16 DTC codes follow.

The sheet is always deleted after sending SEND OBD DTC REQ command.

DTC code must be interpreted this way:
b7………b0   b7………b0
UUUUVVVV CCRRSSSS
(high byte)   (low byte)

CC:	0: digit P	RR:	0: digit 0
	1: digit C		1: digit 1
	2: digit B		2: digit 2
	3: digit U		3: digit 3

SSSS, UUUU,VVVV, value 0..9 corresponds to digits 0..9.

For example, value 0x73C0 is decoded as u0033. 0x2004 as p0420.

SEND CAN USR MSG

```
typedef struct _SPI_USR_SEND_CAN_MESSAGE
{
      LONG id;
      BYTE data[8];
      BYTE len;
      BYTE flags;
      BYTE check[2];
}SPI_USR_SEND_CAN_MESSAGE;
```

`len`	- number of data bytes 0..8
`flags`	- extended identifier 0b00100000 rtr frame 0b01000000
`check`	- check[0]= 0x3F (63d), check[1]= 0xC0 (192d)

SET CAN USR RCV MSG

```
typedef struct _RCV_MSG_TABLE_ITEM
{
      BYTE index;
      LONG id;
      BYTE flags;
      BYTE data[4];
      BYTE check;
}RCV_MSG_TABLE_ITEM;
```

`index` –	index to user accepted CAN messages sheet
`id` -	identifier of the message received into the buffer
`flags` -	b0-standard (0) or extended identifier b1-data (0) or rtr(1) frame b2- if 1 is filtered also according to 0 byte b3- if 1 is filtered also according to 1 byte b4- if 1 is filtered also according to 2 byte b5- if 1 is filtered also according to 3 byte b6- message receiving enabled
`data`	- data bytes that can be used for filtering of received messages
`check`	- controlling byte, value 0x0F (15d)

READ CAN USR RCV MSG

```
typedef struct _READ_CAN_MESSAGE{
      LONG id;
      BYTE data[8];
      BYTE len;
      BYTE flags;
} READ_CAN_MESSAGE;
```

`flags` - extended identifier 0b00100000
rtr frame 0b01000000

Differences in SPI communication in comparison with FMS OEM CHIP V4

1) SETTINGS command allows to set lesser number of CAN speeds. Unused speeds without expected usage were eliminated.
2) Bit reserved by decrease of number of speeds is used for simultaneous receiving of messages with standard and extended identifier.
3) After SETTINGS command, pair of bytes of respective values follow. If not sent, command is ignored.
4) SETTINGS command after start does not activate CAN. It is necessary to use separate command CAN BUS ENABLE or CAN BUS ENABLE LS.
5) SETTINGS2 command is extended to 15 bits. Byte with tachograph type settings follows and 2 controlling bytes as in case of SETTINGS command.
6) Reading of SETTINGS 2 integrated.

RS232 interface

Actual firmware version generates this type of data:

Sign	**ASCII – dec**	**ASCII – hex**	**Description**
R	82	52	Engine rpm
S	83	53	Car speed
A	65	41	Accelerating pedal speed.
B	66	42	Brake pedal speed.
F	70	46	Total fuel consumption value.
H	72	48	Total amount of moto-hours.
L	76	4C	Fuel tank level. In truck – %, VW – litres
C	67	43	Mean consumption.
W *	87	57	Axle weight.
T	84	54	Total number of kilometers on the clock.
D	68	44	Daily number of kilometers.
V	86	56	Distance (km) to the next service control.
N	78	4E	Cooling liquid temperature.
I	73	49	Driver's name on the tachograph card.
K	75	4B	Excess of length of drive without safety stop according to tachograph.
E	69	45	Error codes.
Z	90	5A	Tachograph speed
J	74	4A	Fuel flow
P	80	50	Actual consumption.
U*	85	55	Tire pressure
Q	81	51	Bit states: doors, trunk and so on. 16bit number stated hexadecimally. Bit 0 – closed.

*Command in answer transfers weight (or tire pressure) on more axles. Before each axle weight value, two signs are stated that specify measured weight location. The first sign specifies the axle, the second one wheel position. Axle indexes range is 0-15 and is entered with 0…F signs, it means hexadecimally. F-value means that the position is unknown.

Theoretically, axle weight message can contain weight of 15 axles in this sequence:
W15:0-F-3245:1-F-2252:0-F-1763………E-F-1223
Weight on the first axle (front) is 3.245 kg, measured wheel location is unknown (F), weight of next axle is 2.252 kg, and so on.

Device is able to automatically generate preset data with adjustable generating period 1 s up to 255 minutes.

After start, configuration is read from internal EEPROM. After each change of settings this change is saved to EEPROM.

Ranges of individual quantities:

Data	**Number of numeric signs**	**1 bit value**
Speed	1..3	km/h
Accelerating pedal speed	1..3	%
Brake pedal speed	1..3	%
Total fuel consumption value	1..10	liters
Fuel tank level	1..3	%-truck / liters-car
Engine rpm	1..4	rpm
Axle weight	1..5	kg
Total amount of moto-hours	1..10	h
Total number of kilometers on the clock	1..8	Km
Distance (km) to the next service control	1..8	Km
Cooling liquid temperature	1..3	℃
Mean consumption	1..5	0.1litres / 100 km
Actual consumption	1..5	0.1litres / 100 km
Fuel flow	1..5	0.05 liters/h
Tire pressure	1.4	kPa

Communication examples:

Communication chain contains controlling code. This code is XOR of signs that follow after $ sign to * sign. These signs are not contained in controlling code.

Start
*$PCAN,C,VER,1.10,CANLABsro,*41<enter>*

RS232 device sends one time stated information after the start. Text "1.10" indicates firmware version and this value is changes in each new version. As <enter> sign, these two signs are used CR LF; 0X0D 0x0A (convention Win/DOS).

Firmware version reading
*$PCAN,C,FW,*62<enter>*

Answer
*$PCAN,C,VER,1.10,FMS_CHIP_V5,CANLABsro,*44<enter>*
Firmware version request. Answer has the same format as Start packet.

Data request sent into CAN unit from superordinate system

*$PCAN,C,GET,#Q##,*7B<enter>*

Data about door and cover state requested.

*$PCAN,C,GET,Q0000,*58<enter>*

No doors or cover are open. 4 signs after Q-symbol are hexadecimal number when individual bits represent door state.

For trucks: state is indicated
n state:

Q0000	- all doors are closed
Q0001	- some doors are open

For VW cars:

Q0000	- all doors are closed
Q0001	- driver's door open
Q0002	- co-driver's door open
Q0004	- left rear door open
Q0008	- right rear door open
Q000F	- all 4 doors open
Q0010	- engine bonnet open
Q0020	- baggage compartment bonnet open

Data request sent into CAN unit from superordinate system

*$PCAN,C,GET,#W##,*7D<enter>*

Data about axle weight requested – W-sign, see sheet.

CAN unit answer

*$PCAN,C,GET,W3:1-F-5384:2-F-0:0-F-5343,*19<enter>*

Unit signalizes that weight of 3 axles was measured, returned data signalizes axle location and weight value. Axle No. 1, 2, and 0, data about measured wheel are not available (F), and value in kg. Sign ‘:’ (colon) is used as separator of individual axles.

Alternative answer from CAN unit

*$PCAN,C,GET,WX,*06<enter>*

Unit signalizes that requested data is not available (X-sign after W-sign that indicates data that is not measured – was not accepted from CAN bus).

Data request sent to CAN unit from superordinate system

*$PCAN,C,GET,#SRT##,*7F<enter>*

Data about speed, engine rpm and total number of km on the clock (S, R, and T signs) is requested.

Answer from CAN unit

*$PCAN,C,GET,S16R1313T398405,*58<enter>*

Unit returns data about speed (16 km/h), rpm (1.313 rpm), and kilometer reading (398.405 km).

Alternative answer from CAN unit

*$PCAN,C,GET,SXRXTX,*04<enter>*

Unit returns statement that data is not available. Non-existence of these data is very improbable, thus, it could be expected that CAN line to the unit is interrupted, defect / incorrectly set CAN or car, or unit is active, but car engine is off (CAN of the car is off) – signal error 15.

Periodically generated data according to preset mask have the same form as unit answers.

Data request sent to CAN unit from superordinatate system

*$PCAN,C,GET,#K##,*61<enter>*

Data about excess of length of drive without safety stop requested if this information is transferred from digital tachograph to CAN bus.

*$PCAN,C,GET,K77,*42<enter>*
After the sign, there are 2 hexadecimal codes, each represents state of one tachograph cards. Meaning of the codes:

0 = Rest
1 = Driver available
2 = Work
3 = Drive

6 = Error
7 = Not available

Message with settings to CAN unit 1

*$PCAN,C,SET,C5,EXT,LISO,P10,T0,#SRTL##,*18<enter>*

C5	- CAN-speed settings (usually 5-trucks, 6-engine CAN at VW)
EXT/TWO	- type of CAN-identifier, EXT indicates extended identifier, if EXT is not stated, standard identifier is set (extended identifier – trucks, not stated - standard in case of VW) If “TWO” is stated, both types of identifiers are accepted.
P10	- generate data automatically every 10 minutes, if identifier P is not stated, data are not automatically generated.
LISO	- if identifier is stated, device is working in the mode that ensures that the unit can not influence CAN bus function in the car.
T<n>	- car type, used when two 2 identical identifiers of CAN messages carry different data of different cars, values: 0-trucks; 16-Basic VW ; 48-Basic Mercedes ; 64- Basic Nissan ; 80- Basic Opel ; 96- Basic Toyota
#SRTL##	- between signs # and ##, there is the list of automatically generated data. In this case: S-speed, R-rpm, T-tachometer state, L-fuel tank level.

After receiving of this message, unit processes data, saves it (EEPROM), answers with identical message and proceeds restart. After restart, it continues according to new settings.

*$PCAN,C,SET,C5,EXT,LISO,P0+10,T0,#SRABFHLCTDVNIWU##,*5A<enter>*
Message is generated every 10 minutes

*$PCAN,C,GET,S45R1432A88B0F2428H341L56CXT33457DXVXN93IX,*7B <enter>*

As axle weight and tire pressure is requested (both of them can have 12 signs), two of these messages are generated separately to simplify its parsing. Thus, 3 messages are generated:

*$PCAN,C,GET,S45R1432A88B0F2428H341L56CXT33457DXVXN93IX,*7B <enter>*
*$PCAN,C,GET,W2:0-F-5792:1-F-2055,*66<enter>*
*$PCAN,C,GET,UX,*04<enter>*

Message with settings to CAN unit 2

*$PCAN,C,SET,C5,EXT,LISO,P1+40,T0,#SRTL##,*07<enter>*

This message is the same as above mentioned. The only difference is generating period: set at 1 minute and 40 seconds, which means 100 seconds.

Message with settings to CAN unit 3

*$PCAN,C,SET,C5,EXT,LISO,P0,T0,*3F<enter>*

This message is the same as above mentioned regarding CAN settings. However, periodical messages are not generated. Device provides data upon request.

User data sending to CAN

*$PCAN,C,CAN,S,I123,S,L5,B0:11,B1:22,*3E<enter>*

Message from superordinate system to FMS CHIP. After receiving of this message, CHIP sends message with standard ID 123 with 5 data bytes to CAN. Data bytes values will be 11,22,0,0,0.

User data reading from CAN

*$PCAN,C,CAN,T0,I123,S,B0:11,B1:22,*5C<enter>*

Message from superordinate system sets user buffer 0 for reading CAN to receiving message with standard ID 123. Message must contain identical identifier and value 11 in data byte 0 and value 22 in data byte 1 to be received. If the data byte is stated, it is filtered according to its value. Identifier must be always stated. Filtration can be made according to identifier and optionally according to first 4 data bytes. 8 user buffers are available for receiving: T0 .. T7.

*$PCAN,C,CAN,T1,I1234,E,*7E<enter>*

The message sets user buffer 1 for reading CAN to receiving of the message with expanded identifier. Filtration of receiving to the buffer is made only according to identifier.

*$PCAN,C,CAN,R0,*71<enter>*

Request from superordinate system for value of user buffer 0.

*$PCAN,C,CAN,R0,E,*18<enter>*

Chip answers preceding request for value of user buffer 0 with value E – empty. Buffer responds to message that was not received from CAN or has been already read. Each reading sets buffer to the state “empty” until the message is received from the CAN again.

*$PCAN,C,CAN,R0,I123,S,L8,B0:11,B1:22,B2:3,B3:4,B4:5,B5:6,B6:7,B7:8,*08<enter>*

Chip responds to previous request for value of user buffer 0. Buffer contains message with standard ID 123 with 8 data bytes with values 11,22,3,4,5,6,7,8.

*$PCAN,C,CAN,R1,*70*

Request from superordinate system for value of user buffer 0.

*$PCAN,C,CAN,R1,I1234,E,L8,B0:11,B1:22,B2:3,B3:4,B4:5,B5:6,B6:7,B7:8,*2B*

Chip responds to previous request for value of user buffer 1. Buffer contains message with standard ID 123 with 8 data bytes with values 11,22,3,4,5,6,7,8.

---

*$PCAN,C,CAN,D,*57*

The message deletes all user buffers.

SETTINGS command of the most frequently monitored cars.

Trucks – spinal CAN bus

- listen only
- expanded CAN ID
- speed 250k

$PCAN,C,SET,C5,EXT,LISO,P0+10,T0,#SRTL##,*03<enter>

Truck – FMS gateway

- normal mode
- expanded CAN ID
- speed 250k

$PCAN,C,SET,C5,EXT,P0+10,T0,#SRTL##,*36<enter>

Škoda/VW, engine CAN bus

- listen only
- standard CAN ID
- speed 500k

$PCAN,C,SET,C6,LISO,P0+10,T16,#SRTL##,*52<enter>

In case of VW, FMS OEM CHIP is connected to engine-CAN. Thus, it is not possible to read DTC codes.

Nissan, Renault, Dacia, variant 1 (Nissan Micra)

- listen only
- standard CAN ID
- speed 500k
- experimental support

$PCAN,C,SET,C6,LISO,P0+10,T64,#SRTL##,*57<enter>

Nissan, Renault, Dacia, variant 2

- listen only
- standard CAN ID

- speed 250k
- experimental support

$PCAN,C,SET,C5,LISO,P0+10,T65,#SRTL##,*55<enter>

Ford Mondeo

- listen only
- standard CAN ID
- speed 500k
- experimental support

$PCAN,C,SET,C6,LISO,P0+10,T128,#SRTL##,*6E<enter>

Ford Transit

- listen only
- standard CAN ID
- speed 500k
- experimental support

$PCAN,C,SET,C6,LISO,P0+10,T129,#SRTL##,*6F<enter>

In case of other cars, that support CAN diagnostics, some general data can be read:

- engine temperature
- engine rpm
- car speed
- accelerating pedal
- fuel tank level

Function is in experimental stage.

General OBD car, 11bit ID, 250k

$PCAN,C,SET,C5,P0+10,T1024,#SRTL,*69<enter>

General OBD car, 11bit ID, 500k

$PCAN,C,SET,C6,P0+10,T1024,#SRTL##,*68<enter>

General OBD car, 29bit ID, 250k

$PCAN,C,SET,C5,EXT,P0+10,T1025,#SRTL##,*0D<enter>

General OBD car, 29bit ID, 500k

$PCAN,C,SET,C6, EXT,P0+10,T1025,#SRTL##,*2C<enter>

Car error codes reading

OBD compatible cars allow reading of error codes (DTC). These codes describe errors in the car detected by control units. CAN must be connected to diagnostic connector and LISTEN ONLY mode must be deactivated. If this mode is active, it is possible to deactivate it (preset settings with disabled listen only mode, read car errors and activate it again). The car must support OBD diagnostics through CAN interface.

If OBD diagnostics in this standard form works it depends on car type. However, sending request to the car, that does not support this function, has no influence to the car.

Behind request for DTC errors, 2 parameters follow:

s – standard CAN identifier
e – expanded CAN identifier

Type of identifier depends on car type; the easiest way is to test the standard identifier first and if it does not work, then, try the less frequent one.

3 - diagnostics mode 3, saved error codes
7 - diagnostics mode 7, error codes in actual or previous drive

Function is in experimental stage.

Request 1:
$PCAN,C,GET,#E<s3>##,*2D<enter>

Answer:
$PCAN,C,GET,E:P0107P0113,*73<enter>
2 error codes are returned: P0107 and P0113.
Meaning of the codes can be found here: http://www.obd-codes.com/ .
P0107 - Manifold Absolute Pressure/Barometric Pressure Circuit Low Input
P0113 - Intake Air Temperature Circuit High Input

Request2:
$PCAN,C,GET,#E<s3>##,*2D

Answer:
```
$PCAN,C,GET,E:U0073P0420,*71<enter>
$PCAN,C,GET,E:U0101U0155C0444,*30<enter>
$PCAN,C,GET,E:C0534C0446C2647C3700,*71<enter>
$PCAN,C,GET,E:P0011P0012,*75<enter>
```

In this case, several answers were returned. The reason is that some control unit contains many error codes or several control unit answers.
Error codes: U0073 P0420 U0101 U0155 C0444 C0534 C0446 C2647 C3700 P0011 P0012.

Power control

Customer can optionally order the chip with POWER CONTROL function. In this case, RE0 pin is designed to connection to ignition signal or other control signal. RE2 pin controls the supply. Respective supply and connection scheme is available upon request.

Superordinate system can switch on FMS OEM CHIP by means of MCLR pin.

Order specification

It is necessary to specify several parameters in the order:

1) If the chip is in SPI mode, positive or negative CS can be chosen. Negative CS by default, thus, chip communicates, if CS is low.
2) If the chip is used with SPI at low CLK speed (e.g. with GSM/GPS modules Telit through GPIO), it is necessary to specify it.
3) Function Power control switched on or off.
4) Active (settings are saved after change to EEPROM) / inactive CAN when it is necessary to set the device after each start (preferred).
5) It is possible to adjust chip functioning upon request, e.g. preset specific settings during its programming.

Warning

Customer attaches the chip to the car at his own risk. Incorrect setting of the chip can cause incorrect function of car control units. CANLAB s.r.o. is not liable for any damage of the car.