ETH2CAN v3 – BLACKBOX

Obsah:

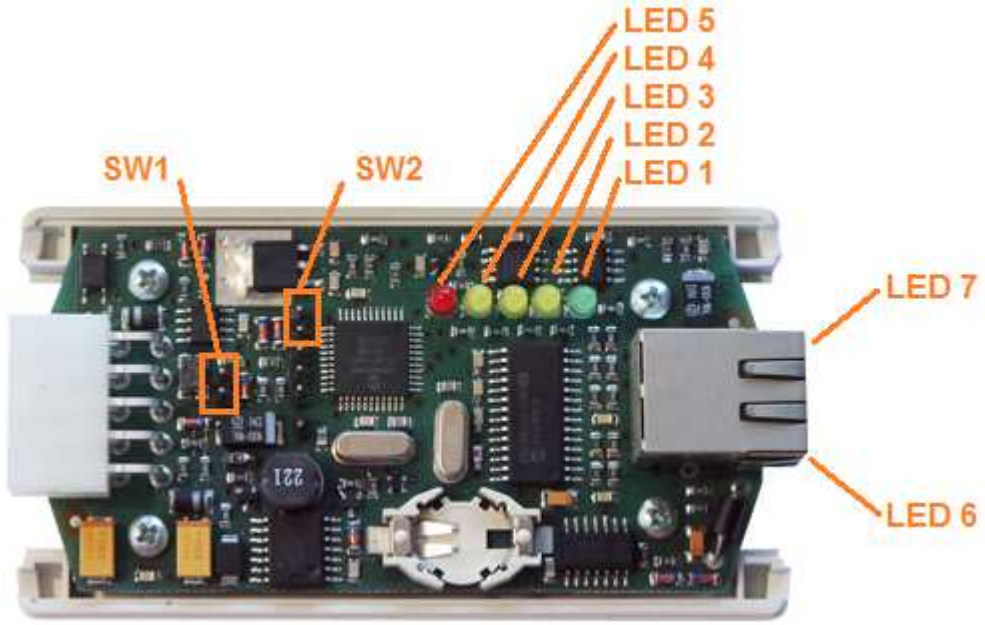
Ing. David Španěl

Mgr. Vítězslav Rejda

CANLAB s.r.o.

Základní popis

Interface ETH2CAN model 3 (blackbox) je určen pro připojení ke sběrnici CAN prostřednictvím sítě ethernet. Rychlost ethernetového rozhraní je 10Mbitu. Zařízení zpracovává data ze sběrnice CAN a ty zpřístupňuje pomocí ethernetového rozhraní a protokolu TCP/IP informačnímu systému vozidla. Zároveň ukládá data do interní paměti odkud je možné zpětně vyčíst průběh jízdy. Data jsou ukládána do dvou oddělených pamětí.
První paměť označovaná jako FRAM ukládá některá jízdní data při příchodu dráhového impulzu. Tato paměť obsahuje data za posledních 500m jízdy s vysokým rozlišením. Druhá obsahuje data ukládané na základě nastavené periody záznamu. Period je volitelná v rozsahu stovek milisekund do desítek sekund.



Zařízení ETH2CAN ve verzi 3 obsahuje:

- rozhranní pro připojení sběrnice CAN
- rozhranní ETHERNET pro připojení k informačnímu systému
- zapínání signálem 15
- vstup pro dráhové impulzy
- možnost osadit a využít ve firmware 1 digitální vstup
- možnost osadit a využít ve firmware 1 digitální výstup s galvanickým oddělením

Komunikace po rozhraní ETHERNET

Zařízení má svoji IP adresu a TCP port prostřednictvím kterého probíhá veškerá komunikace. Chová se jako server, tedy klient se připojuje k tomuto zařízení. Jsou podporována maximálně 4 paralelní připojení.

Pro komunikaci se používá několika paketů, každý paket obsahuje položku magic, ta je určena k ověření endianu zařízení. Má hodnotu 0xAA123456. Dále pak položku packet_type, ta rozlišuje typ paketu. Položka packet_size pak určuje velikost paketu. Položka id je určena k identifikaci paketu. Je li například zaslán do zařízení dotaz, je možné nastavit položku id na libovolnou hodnotu. Paket s odpovědí pak má nastavenou stejnou hodnotu. Lze tak rozlišit dvě odpovědi od sebe při zaslání dvou požadavků.

Typy paketů:

```
#define   PACKET_RUN                    1
#define   PACKET_MODE                   3
#define   PACKET_RESET                  4
#define   PACKET_CLOSE                  16
#define   PACKET_WAYDATA                32
#define   PACKET_TIMEDATA               33
#define   PACKET_INPUTS                 34
#define   PACKET_OUTPUTS                35
#define   PACKET_TIMEDATA2              36 *
#define   PACKET_VEHICLE_TYPE           40 ****
#define   PACKET_EXTENDED_TIMEDATA      48 **
#define   PACKET_STATUS                 128
#define   PACKET_REBOOT_DATA            129
#define   PACKET_BOOT_LOCK              130
#define   PACKET_DEBUG_1                131
#define   PACKET_GET_RTC                160
#define   PACKET_SET_RTC                161
#define   PACKET_READ_FRAM              164
#define   PACKET_ERASE_FRAM             165
#define   PACKET_DISABLE_FRAM           166
#define   PACKET_READ_FLASH             172
#define   PACKET_ERASE_FLASH            173
#define   PACKET_SETTINGS_UNLOCK        200***
#define   PACKET_UNLOCK_STATUS          201***
#define   PACKET_FULL_SETTINGS_EEPROM   251
```

```
#define   PACKET_FULL_SETTINGS              252
#define   PACKET_SERIAL_NUMBER              253
#define   PACKET_FIRMWARE_VERSION           254
#define   PACKET_CONFIGURATION              255
```

* Od verze FW 2.00 nahrazuje paket PACKET_TIMEDATA2 předchozí verzi PACKET_TIMEDATA.
** Od verze FW 3.00 doplněn PACKET_ EXTENDED_TIMEDATA.
*** Od verze FW 3.02
**** Od verze FW 4.00

Podpora paketů:

Paket	Bootloader	Aplikace	Perioda
PACKET_RUN	Y	Y	async
PACKET_MODE	Y	Y	async
PACKET_RESET	Y	Y	async
PACKET_CLOSE	N	Y	async
PACKET_WAYDATA	N	Y	~200ms
PACKET_TIMEDATA / PACKET_TIMEDATA2	N	Y	~500ms
PACKET_EXTENDED_TIMEDATA	N	Y	~1000ms
PACKET_INPUTS	N	Y	async
PACKET_OUTPUTS	N	Y	async
PACKET_STATUS	N	Y	async
PACKET_REBOOT_DATA	Y	N	async
PACKET_GET_RTC	N	Y	async
PACKET_SET_RTC	N	Y	async
PACKET_READ_FRAM	N	Y	async
PACKET_ERASE_FRAM	N	Y	async
PACKET_DISABLE_FRAM	N	Y	async
PACKET_READ_FLASH	N	Y	async
PACKET_ERASE_FLASH	N	Y	async
PACKET_FULL_SETTINGS_EEPROM	N	Y	async
PACKET_FULL_SETTINGS	N	Y	async
PACKET_FIRMWARE_VERSION	Y	Y	async
PACKET_SETTINGS	N	Y	async

Struktury paketů je třeba zarovnat na 1 bajt. Tedy bez jakékoliv další výplně.
V jazyce C/C++ v prostředí Microsoft Visual Studia je toho možné dosáhnout například takto:

```
#pragma pack(push, 1)
typedef struct _ETH_HEADER {
          unsigned __int32    magic;
          unsigned char       packet_type;
          unsigned __int16    packet_size;
          unsigned char       id;
} ETH_HEADER;
#pragma pack(pop)
```

Paket RUN

Paket ve směru Klient (nadřazený systém) -> ETH2CAN.

```
typedef struct _ETH_RUN {
     unsigned __int32    magic;
     unsigned char       packet_type;
     unsigned __int16    packet_size;
     unsigned char       id;
}    ETH_RUN;
```

Paket je určen k aktivaci zařízení. Po připojení zařízení k napájení nacházzí v režimu bootloaderu. Tento režim je určen pro snadnou změnu firmware v zařízení. Zasláním tohoto paketu dojde k aktivaci firmware. Bootloader je automaticky přepnut do režimu firmware po uplynutí intervalu 1 sekundy, pokud bootloader nedetekuje příchod paketu PACKET_REBOOT_DATA.

Paket ve směru ETH2CAN -> Klient (nadřazený systém).

```
typedef struct _ETH_RUN2 {
     unsigned __int32    magic;
     unsigned char       packet_type;
     unsigned __int16    packet_size;
     unsigned char       id;
     unsigned char       mode;
}    ETH_RUN2;
```

Paket je generován jako odpověď na příchozí paket ETH_RUN. Potvrzuje přijetí tohoto paketu a zároveň v položce mode signalizuje aktuální režim firmware (bootloader: mode = 1, application-firmware: mode = 2)

Paket MODE

Paket ve směru Klient (nadřazený systém) -> ETH2CAN.

```
typedef struct _ETH_MODE {
     unsigned __int32    magic;
     unsigned char       packet_type;
     unsigned __int16    packet_size;
     unsigned char       id;
}    ETH_MODE;
```

Paket je určen k vyžádání aktuálního režimu firmware (bootloader/aplikace).

Paket ve směru ETH2CAN -> Klient (nadřazený systém).

```
typedef struct _ETH_MODE2 {
     unsigned __int32    magic;
```

```
     unsigned char       packet_type;
     unsigned __int16    packet_size;
     unsigned char       id;
     unsigned char       mode;
}    ETH_MODE2;
```

Paket je generován jako odpověď na příchozí paket ETH_MODE. Potvrzuje přijetí tohoto paketu a zároveň v položce mode signalizuje aktuální režim firmware (bootloader: mode = 1, application: mode = 2).

Paket RESET

```
typedef struct _ETH_RESET {
     unsigned __int32    magic;
     unsigned char       packet_type;
     unsigned __int16    packet_size;
     unsigned char       id;
}    ETH_RESET;
```

Paket ve směru Klient (nadřazený systém) -> ETH2CAN. Po přijetí tohoto paketu dojde k restartu firmware řídicího procesoru a spuštění režimu bootloaderu.

Paket CLOSE

```
typedef struct _ETH_SHUTDOWN {
     unsigned __int32    magic;
     unsigned char       packet_type;
     unsigned __int16    packet_size;
     unsigned char       id;
}    ETH_RESET;
```

Paket ve směru Klient (nadřazený systém) -> ETH2CAN. Nadřazený systém indikuje ETH2CANu ukončení spojení.

Pakety PACKET_WAYDATA

```
typedef struct _ETH_WAYDATA {
     unsigned __int32    magic;
     unsigned char       packet_type;
     unsigned __int16    packet_size;
     unsigned char       id;

     unsigned char       packet_counter;          //1
     unsigned __int32    pulses;                  //5
     ////////////////////////////////////////
```

```
unsigned __int16    rychlost;               //7
////////////////////////////////////////
unsigned char       provozni_brzda      : 1; //8
unsigned char       spojka              : 1;
unsigned char       low_idle            : 1;
unsigned char       kick_down           : 1;
unsigned char       parkovaci_brzda     : 1;
unsigned char       zastavkova_brzda    : 1;
unsigned char       neutral             : 1;
unsigned char       zpatecka            : 1;
////////////////////////////////////////
unsigned char       beh_motoru          : 1; //9
unsigned char       ind_tlaku_oleje_motoru : 1;
unsigned char       ind_tlaku_vzduchu   : 1;
unsigned char       alternator_1        : 1;
unsigned char       alternator_2        : 1;
unsigned char       aktiv_retarderu     : 1;
unsigned char       aktiv_tempomat      : 1;
unsigned char       sv_obrysova         : 1;
////////////////////////////////////////
unsigned char       sv_tlumena          : 1; //10
unsigned char       sv_dalkova          : 1;
unsigned char       smer_leva           : 1;
unsigned char       smer_prava          : 1;
unsigned char       osvet_prost_ridice  : 1;
unsigned char       osvet_prost_cestujicich  : 1;
unsigned char       sign_cestujicich    : 1;
unsigned char       sign_kocarek        : 1;
////////////////////////////////////////
unsigned char       sign_invalida       : 1; //11
unsigned char       sign_zastavime      : 1;
unsigned char       uvol_samoobs_dveri  : 1;
unsigned char       ochrana_dveri_secu  : 1;
unsigned char       tl_dvere_1          : 1;
unsigned char       dvere_1_otevreny    : 1;
unsigned char       tl_dvere_2          : 1;
unsigned char       dvere_2_otevreny    : 1;
////////////////////////////////////////
unsigned char       tl_dvere_3          : 1; //12
unsigned char       dvere_3_otevreny    : 1;
unsigned char       tl_dvere_4          : 1;
unsigned char       dvere_4_otevreny    : 1;
unsigned char       tl_dvere_5          : 1;
unsigned char       dvere_5_otevreny    : 1;
unsigned char       plosina_vysunuta    : 1;
unsigned char       pridavne_topeni     : 1;
////////////////////////////////////////
unsigned char       ind_obd             : 1; //13
unsigned char       res_1               : 1;
unsigned char       res_2               : 1;
unsigned char       res_3               : 1;
```

```
    unsigned char       rezim_kr_mom        : 4;
    ////////////////////////////////////////
    unsigned char       dummy;    //14 // zarovnani na sudy
                                        // pocet
} ETH_WAYDATA;
```

Paket PACKET_TIMEDATA

```
typedef struct _TIME_RECORD_1
{
    unsigned char       aktual_kr_mom_motoru;    //1
    unsigned __int16    rpm;                     //3
    unsigned char       acc_pedal;               //4
    unsigned __int16    okam_spotreba;           //6
    unsigned char       plnici_tlak;             //7
    unsigned char       tlak_oleje_motoru;       //8
    unsigned char       prevodovy_stupen;        //9
    unsigned __int16    tlak_brzdovy_1;          //11
    unsigned char       dummy;                   //12
} TIME_RECORD_1;

typedef struct _TIME_RECORD_2
{
    unsigned char       aktual_kr_mom_retarderu; //1
    unsigned __int16    napeti_baterie;          //3
    unsigned char       teplota_chladiva_motoru; //4
    unsigned __int16    teplota_oleje_prevodovky;//6
    unsigned __int16    teplota_oleje_retarderu; //8
    unsigned char       teplota_vzduchu_sani;    //9
    unsigned __int16    tlak_brzdovy_2;          //11
    unsigned char       dummy;
} TIME_RECORD_2;

typedef union _TIME_RECORD_3
{
    unsigned __int32    celkove_km;              //4
    unsigned __int32    celkove_palivo;          //8
    unsigned char       zasoba_paliva;           //9
    unsigned char       ridic_1             : 3;
    unsigned char       ridic_2             : 3;
    unsigned char       detekce_pohybu : 2;      //10
    unsigned char       datum_cas[6];            //16
} TIME_RECORD_3;

typedef struct _TIME_RECORD_4
{
    unsigned char       location1;
    unsigned __int16    weight1;
    unsigned char       location2;
    unsigned __int16    weight2;
```

```
} TIME_RECORD_4;

typedef struct _ETH_TIMEDATA {
     unsigned __int32    magic;
     unsigned char       packet_type;
     unsigned __int16    packet_size;
     unsigned char       id;

     unsigned __int16    packet_counter;
     TIME_RECORD_1       tr1;
     TIME_RECORD_2       tr2;
     TIME_RECORD_3       tr3;
     TIME_RECORD_4       tr4;
} ETH_TIMEDATA;
```

Od verze FW 2.00 je paket nahrazen novou variantou. Struktura `TIME_RECORD_4` je nahrazena verzí `TIME_RECORD_5.`

```
typedef struct _ETH_TIMEDATA2 {
     unsigned __int32    magic;
     unsigned char       packet_type;
     unsigned __int16    packet_size;
     unsigned char       id;

     unsigned __int16    packet_counter;
     TIME_RECORD_1       tr1;
     TIME_RECORD_2       tr2;
     TIME_RECORD_3       tr3;
     TIME_RECORD_5       tr5;
} ETH_TIMEDATA2;


typedef struct _TIME_RECORD_5
{
     unsigned char location1;
          unsigned char ecu1;
          unsigned __int16  weight1;

          unsigned char location2;
          unsigned char ecu2;
          unsigned __int16  weight2;

          unsigned char location3;
          unsigned char ecu3;
          unsigned __int16  weight3;
} TIME_RECORD_5;

typedef struct _ PACKET_EXTENDED_TIMEDATA
{
     unsigned __int16    packet_counter;
```

```
    unsigned __int16    wheel_speed;
    unsigned char       ETC1;
    unsigned __int16    ASC3_ecu1[4];
    unsigned char       ASC3_ecu1_addr;
    unsigned char       ASC1_ecu1[8];
    unsigned char       ASC1_ecu1_addr;
    unsigned int        ASC3_ecu2[4];
    unsigned char       ASC3_ecu2_addr;
    unsigned char       ASC1_ecu2[8];
    unsigned char       ASC1_ecu2_addr;
    unsigned char       dummy;
} PACKET_EXTENDED_TIMEDATA;
```

Data ASC1 a ASC2 je možné monitorovat ze 2 ECU. V ecu 1 jsou uloženy data z první přijaté zprávy. V ecu 2 se ukládá od druhé případně 3 a další ecu. Je li ecu více než 2, údaj v ecu 2 se střídá podle posledního přijatého údaje.

Air Suspension Control 3 - ASC3

Start Position	Length	Parameter Name
0-1	2 bytes	Relative Level Front Axle Left
2-3	2 bytes	Relative Level Front Axle Right
4-5	2 bytes	Relative Level Rear Axle Left
6-7	2 bytes	Relative Level Rear Axle Right

Resolution: 0.1 mm/bit, -3,200 mm offset
Data Range: -3,200 to 3,225.5 mm

Air Suspension Control 1 - ASC1

Start Position	Length	Parameter Name
1.1	4 bits	Nominal Level Front Axle
1.5	4 bits	Nominal Level Rear Axle
2.1	2 bits	Below Nominal Level Front Axle
2.3	2 bits	Below Nominal Level Rear Axle
2.5	2 bits	Above Nominal Level Front Axle
2.7	2 bits	Above Nominal Level Rear Axle
3.1	2 bits	Lowering Control Mode Front Axle
3.3	2 bits	Lowering Control Mode Rear Axle
3.5	2 bits	Lifting Control Mode Front Axle
3.7	2 bits	Lifting Control Mode Rear Axle
4.1	4 bits	Kneeling Information
4.5	4 bits	Level Control Mode
5.1	2 bits	Security Device
5.3	2 bits	Vehicle Motion Inhibit
5.5	2 bits	Door Release
5.7	2 bits	Lift Axle 1 Position
6.1	2 bits	Front Axle in Bumper Range
6.3	2 bits	Rear Axle in Bumper Range
6.7	2 bits	Lift Axle 2 Position
7.1	2 bits	Suspension Remote Control

Nominal Level
Signal which indicates the nominal (desired) height of the axle to be controlled by the suspension system.
These heights are discrete levels. They are the upper level, lower level, normal level 1, normal level 2, normal level 3, customer level, and preset level.
· Upper Level is the highest mechanically available height of the vehicle.
· Lower Level is the lowest mechanically available height of the vehicle.

0000 Level not specified,(i.e. the nominal level is none of the specified levels, no error condition)
0001 "Normal Level 1,(i.e. the level prescribed for normal driving, given by design)
0010 "Normal Level 2,(i.e. a level permitted for driving, for example to lower the vehicle in case of high speed)
0011 "Normal Level 3,(i.e. a level permitted for driving, for example to lift the vehicle in case of offroad)

0100 "Preset Level,(i.e. a level to be defined externally via CAN)
0101 "Customer Level,(i.e. a level to be defined by customer via parameter setting)
0110 "Upper Level,(i.e. the highest level to be controlled)
0111 "Lower Level,(i.e. the lowest level to be controlled)
1000-1101 Not defined
1110 Error
1111 Not available

Below Nominal Level

Signal which indicates whether the actual height of the front axle is below the nominal (desired) leve. For explanations of nominal level see parameter - Nominal Level

Above Nominal Level Front Axle

Signal which indicates whether the actual height of the axle is above the nominal (desired) level of the axle. For explanations of nominal level see Nominal Level.

Lowering Control Mode Front Axle

Signal which indicates the actual lowering level change at the front axle
00 Lowering not active
01 Lowering active
10 Error
11 Not available

Lifting Control Mode Front Axle

Signal which indicates the actual lifting level change at the front axle
00 Lifting not active
01 Lifting active
10 Error
11 Not available

Kneeling Information

Signal which indicates the actual level change in case of kneeling function
0000 Not active,(i.e. the kneeling function is not active")
0001Lowering active,(i.e. the vehicle is lowered due to a kneeling request)
0010 Kneeling level reached,(i.e. the vehicle is at the fixed kneeling level)
0011 Lifting active,(i.e. the vehicle is lifted due to a recover request)
0100 Kneeling aborted,(i.e. in case of manual actuation the request was dropped before the kneeling level was reached)
0101-1101 Not defined
1110 Error
1111 Not available

Level Control Mode

Signal which indicates the actual control mode of the air suspension system
0000 Normal operation; i.e. the system performs a ""pure"" control of the vehicle height
0001 Traction help (load transfer); i.e. the driven axle is loaded to a maximum value given by legislation or design
0010 Load fixing; i.e. the driven axlen is loaded to a value defined by the driver
0011 Pressure ratio 1; i.e. the ratio between the pressures at the driven axle and at the third axle is controlled, so that the ratio equals a fixed value 1
0100 Pressure ratio 2; i.e. the ratio between the pressures at the driven axle and at the third axle is controlled, so that the ratio equals a fixed value 2
0101 Optimum traction 1; i.e. the pressure at the driven axle is controlled at a fixed value 1
0110 Optimum traction 2; i.e. the pressure at the driven axle is controlled at a fixed value 2
0111 Traction help - load reduce; (i.e. the driven axle load is reduced to normal load condition)
1000 Exhausting bellow function; i.e. the bellows are exhausted totally
1001-1101 Not defined
1110 Error
1111 Not available

Security Device

The signal which indicates the status of the security device. An example of a security device is a curbstone feeler installed beneath the doors of a bus. If the security device becomes active during kneeling the kneeling process (lowering) is stopped and the vehicle lifts back to the starting level.
00 Not active
01 Active
10 Error
11 Not available

Vehicle Motion Inhibit

Signal which indicates whether vehicle motion is inhibited.
00 Vehicle may be moved
01 Vehicle motion is inhibited
10 Error

11 Not available

Door Release
Signal which indicates that the doors may be opened. [Please note: doors instead of door!]
In case a kneeling request is active the ASC indicates during lowering the vehicle „doors shall not be opened“ as a security information until the kneeling level is reached. Then "doors may be opened" is sent.
00 Doors may not be opened
01 Doors may be opened
10 Error
11 Not available

Lift Axle 1 Position
Signal which indicates the position / load condition of lift axle / tag axle #1. Numbering of lift/tag axles starts at front axle.
00 Lift axle position down / tag axle laden
01 Lift axle position up / tag axle unladen
10 Error

Front Axle in Bumper Range
Signal which indicates that the vehicle height at the front axle (SPNs 1721 and 1722) is within the bumper range.
00 Actual level out of bumper range
01 Actual level within bumper range
10 Error
11 Not available

Suspension Remote Control 1
Signal which indicates that the suspension system is controlled by remote control #1. Remote control is an external unit to operate the suspension system.
00 Not active
01 Active
10 Error
11 Not available

Suspension Control Refusal Information
Signal which indicates that the air suspension control cannot perform a request due to the operating conditions. It also provides a reason for the refusal.
0000 Actual request not refused
0001 Axle load limit reached (load transfer)
0010 Would exceed axle load limit (tag axle)
0011 Bogie differential not locked
0100 Above speed limit
0101 Below speed limit
0110 General reject; i.e. no specified reason applies
0111 - 1101 Not defined
1110 Error
1111 Not available

Paket PACKET_INPUTS

Paket ve směru Klient (nadřazený systém) -> ETH2CAN. Dotaz na stav digitálních vstupu.

```
typedef struct _ETH_INPUTS {
     unsigned __int32    magic;
     unsigned char       packet_type;
     unsigned __int16    packet_size;
     unsigned char       id;
}    ETH_INPUTS;
```

Paket ve směru ETH2CAN -> Klient (nadřazený systém).

```
typedef struct _ETH_INPUTS2 {
     unsigned __int32    magic;
     unsigned char       packet_type;
     unsigned __int16    packet_size;
     unsigned char       id;
```

```
     unsigned char       inputs;
}    ETH_INPUTS2;
```

Inputs:
bit 0 – univerzální vstup
bit 1 – stav vstupu dráhových pulzů
bit 2 – stav vstupu zapalování (key, „15“)
bit 3 – stav vstupu pro nastavení defaultního nastavení převodníku ETH2CAN

Paket PACKET_OUTPUTS

Paket ve směru Klient (nadřazený systém) -> ETH2CAN. Dotaz na stav digitálních výstupů.

```
typedef struct _ETH_OUTPUTS {
     unsigned __int32    magic;
     unsigned char       packet_type;
     unsigned __int16    packet_size;
     unsigned char       id;
}    ETH_INPUTS;
```

Paket ve směru ETH2CAN -> Klient (nadřazený systém) = stav digitálních výstupů nebo ve směru Klient (nadřazený systém) -> ETH2CAN = nastavení digitálních výstupů.

```
typedef struct _ETH_INPUTS2 {
     unsigned __int32    magic;
     unsigned char       packet_type;
     unsigned __int16    packet_size;
     unsigned char       id;
     unsigned char       outputs;
}    ETH_INPUTS2;
```

Outputs:
bit 0 – univerzální výstup

Pakety PACKET_GET_RTC a PACKET_SET_RTC

```
typedef struct _ETH_GET_RTC {
     unsigned __int32    magic;
     unsigned char       packet_type;
     unsigned __int16    packet_size;
     unsigned char       id;

     unsigned char       month;
     unsigned char       year;
     unsigned char       hour;
     unsigned char       minute;
```

```
    unsigned char       second;
} ETH_GET_RTC;
```

Paket PACKET_READ_FRAM

Paket ve směru Klient (nadřazený systém) -> ETH2CAN. Požadavek na čtení dat z paměti FRAM (64kB).

```
typedef struct _ETH_READ_FRAM {
    unsigned __int32    magic;
    unsigned char       packet_type;
    unsigned __int16    packet_size;
    unsigned char       id;

    unsigned long       address;
    unsigned char       size;

} ETH_READ_FRAM;
```

V paměti FRAM jsou data uloženy sekvenčně do paměti bezprostředně za sebou od adresy 0 takto:

```
unsigned char       date[3];
unsigned char       time[3];

// nasleduji polozky bez hlavicek
PACKET_WAYDATA
PACKET_TIMEDATA     – jen TIME_RECORD_1 .. TIME_RECORD_3,
                    TIME_RECORD_1 není uveden
```

Doporučovaný způsob čtení je tedy tak, aby bylo možné příchozí paket identifikovat touto strukturou:

```
typedef struct _ETH_READ_FRAM {
    unsigned __int32    magic;
    unsigned char       packet_type;
    unsigned __int16    packet_size;
    unsigned char       id;

    unsigned long       address;
    unsigned char       size;

    unsigned char       date[3];
    unsigned char       time[3];

    // nasleduji polozky bez hlavicek
    PACKET_WAYDATA
    PACKET_TIMEDATA

} ETH_READ_FRAM;
```

Položka `size` (velikost bloku čtené paměti) pak má velikost:

```
size = 3 + 3 + sizeof(PACKET_WAYDATA)+
sizeof(PACKET_TIMEDATA);
```

a odpovídá přesně velikosti jednoho záznamu. Čtení většího bloku dat může způsobit přetečení interního bufferu.

Paket PACKET_DISABLE_FRAM

Paket ve směru Klient (nadřazený systém) -> ETH2CAN. Požadavek na čtení stavu blokace fram.

```
typedef struct _ETH_RESET {
    unsigned __int32    magic;
    unsigned char       packet_type;
    unsigned __int16    packet_size;
    unsigned char       id;
}   ETH_RESET;
```

Paket ve směru Klient (nadřazený systém) -> ETH2CAN: požadavek na blokování nebo odblokování zápisu do FRAM.

Hodnota disable:
170d - blokování zápisu
240d – odblokování zápisu
Ostatní hodnoty ignorovány.

```
typedef struct _ETH_RESET {
    unsigned __int32    magic;
    unsigned char       packet_type;
    unsigned __int16    packet_size;
    unsigned char       id;
    unsigned char       disable;
}   ETH_RESET;
```

Paket ve směru ETH2CAN -> Klient (nadřazený systém): odpověď na čtení stavu blokování zápisu do FRAM.

```
typedef struct _ETH_RESET {
    unsigned __int32    magic;
    unsigned char       packet_type;
    unsigned __int16    packet_size;
    unsigned char       id;
    unsigned char       disable;
}   ETH_RESET;
```

170d - zápis blokování
0d nebo 240d – zápis povolen

Paket PACKET_READ_FLASH

Paměť FLASH obsahuje záznamy jízdních parametrů ukládaných s nastavenou periodou. Obsahuje tak data za řádově větší dobu jízdy než paměť FRAM. Protože použitá paměť pracuje s nejmenším mazatelným blokem o velikosti 4kB, je záznam organizován do bloků s tou to velikostí.

Každý tento blok obsahuje na začátku údaj (hlavička bloku) o době jeho vzniku a o tom zda jde o navazující blok na blok předchozí nebo o nový záznam – nová jízda. Algoritmus hledání volného/nejstaršího bloku pracuje tak, že blok je označen časovou značkou vypočtenou jako počet sekund od 0 hodin, 0 minut, 0 sekund 1.1.2000. Protože jnterně dochází pouze k porovnání hodnoty, nikoliv ke zpětnému vypočítávání data, je výpočet zjednodušen tak, že se předpokládá že měsíc může mít 31 dnů.

```
typedef struct _FLASH_SECTOR_MARK {
     unsigned long       type; // 0 - begin, 1 .... next
     unsigned long       time;
     unsigned char       res[4];
     unsigned char       rtc_date[3];
     unsigned char       rtc_time[3];
} FLASH_SECTOR_MARK;
```

Vlastní záznamy začínají na offsetu 96d od začátku sektoru. Každý záznam má stejnou strukturu jako záznam v FRAM. Tedy takto:

```
     unsigned char       date[3];
     unsigned char       time[3];

     // nasleduji polozky bez hlavicek
     PACKET_WAYDATA
     PACKET_TIMEDATA     - včetně TIME_RECORD_4
```

Neleží však přímo za sebou ale jsou uloženy v rastru 80-ti bajtů. Rozdíl mezi velikostí dat a rastru 80 bajtů je ponechán jako rezerva pro možnosti uložení dalších dat při zachování zpětné kompatibility.

```
typedef struct _ETH_READ_FLASH {
     unsigned __int32    magic;
     unsigned char       packet_type;
     unsigned __int16    packet_size;
     unsigned char       id;

     unsigned long       address;
     unsigned char       size;

     unsigned char       date[3];
     unsigned char       time[3];
```

```
    // nasleduji polozky bez hlavicek
    PACKET_WAYDATA
    PACKET_TIMEDATA

} ETH_READ_FLASH;
```

Doporučený způsob čtení FLASH je přečtení všech hlaviče a následně čtení jednotlivých položek uživatelem dle výběru z hlaviček.

Od verze FW 2.00 je položka `PACKET_TIMEDATA` nahrazena verzí `PACKET_TIMEDATA20`. Prokládací rastr je zmenšen o velikost struktur z 80 bajtů na 74.

Paket PACKET_ERASE_FLASH a PACKET_ERASE_FRAM

Příkaz provede vymazání obsahu paměti FLASH a dojde k restartu zařízení. Příkaz má význam pouze pro účely testování firmware.

Pakety PACKET_SETTINGS_UNLOCK a PACKET_UNLOCK_STATUS

Od verze FW 3.02 došlo k úpravě mechanismu konfigurace zařízení. Z bezpečnostních důvodů byl doplněn mechanismus odemčení přístupu ke konfiguraci. Pro povolení konfigurace je nutno před zasláním paketu SETTINGS nebo FULL_SETINGS odemknout přístup do konfigurační paměti. Tento přístup je povolen na 30 sekund od zaslání paketu PACKET_SETTINGS_UNLOCK. Po uplynutí tohoto času je přístup opět uzamčen. Paket PACKET_SETTINGS_UNLOCK obsahuje hlavičku následovanou 4 bajtovým slovem s hodnotou 0xA1B1C1D1. Potvrzení přijetí tohoto požadavku je potvrzeno odpovědí s typem paketu PACKET_UNLOCK_STATUS, kdy za hlavičkou následuje jeden datový bajt s hodnotou 0x01.

V případě zaslání paketu SETTINGS nebo FULL_SETINGS bez předchozího odemčení nebo po uplynutí 30 sekundového intervalu jsou tyto pakety potvrzeny chybou paketem PACKET_UNLOCK_STATUS , kdy za hlavičkou následuje jeden datový bajt s hodnotou 0x00.

Paket SETTINGS

Paket ve směru Klient (nadřazený systém) ->ETH2CAN. Nastavuje novou konfiguraci zařízení. Zařízení odpovídá zasláním tohoto paketu zpět.

```
typedef struct {
    unsigned __int32    magic;
    unsigned char       packet_type;
```

```
     unsigned __int16    packet_size;
     unsigned char       id;
     unsigned char       can_speed;
     unsigned char       listen_only;
     unsigned char       app_mode;
} ETH2CAN_SETTINGS;
```

Paket ve směru ETH2CAN -> Klient (nadřazený systém).
Slouží k zjištění aktuální konfigurace zařízení:

```
typedef struct _ETH2CAN_SETTINGS_REQ {
     unsigned __int32    magic;
     unsigned char       packet_type;
     unsigned __int16    packet_size;
     unsigned char       id;
}    ETH2CAN_SETTINGS_REQ;
```

can_speed – rychlost CAN sběrnice, hodnoty

0	10k
1	20k
2	33.3k
3	50k
4	62.5k
5	83.3k

6	100k
7	125k
8	250k
9	500k
10	1M

listen_only
0 normální mód (připojení na FMS bránu)
1 listen only mód (připojení na CAN bus vozidla, motorový CAN)

app_mode
položka určena pro budoucí použití

Paket PACKET_FULL_SETTINGS

Paket ve směru Klient (nadřazený systém) ->ETH2CAN. Nastavuje novou plnou konfiguraci zařízení. Zařízení odpovídá zasláním tohoto paketu zpět.

```
typedef struct {
     unsigned __int32    magic;
     unsigned char       packet_type;
     unsigned __int16    packet_size;
     unsigned char       id;
     unsigned char       can_speed;
     unsigned char       listen_only;
     unsigned char       app_mode;
     unsigned char       ip[4];
     unsigned char       mask[4];
     unsigned char       mac[6];
```

```
    unsigned __int16    port;
    unsigned char       max_connections;
    unsigned __int16    flash_period;
    unsigned char       app_start_delay;

} ETH2CAN_SETTINGS;
```

Paket ve směru ETH2CAN -> Klient (nadřazený systém).
Slouží k zjištění aktuální plne konfigurace zařízení:

```
typedef struct _ETH2CAN_SETTINGS_REQ {
    unsigned __int32    magic;
    unsigned char       packet_type;
    unsigned __int16    packet_size;
    unsigned char       id;
}   ETH2CAN_SETTINGS_REQ;
```

can_speed – rychlost CAN sběrnice, hodnoty

0	10k
1	20k
2	33.3k
3	50k
4	62.5k
5	83.3k

6	100k
7	125k
8	250k
9	500k
10	1M

listen_only
0 normální mód (připojení na FMS bránu)
1 listen only mód (připojení na CAN bus vozidla, motorový CAN)

app_mode
položka určena pro budoucí použití

ip
IP adresa zařízení. Defaultně přednastavena na 192.168.12.150. Je možné však vyžádat z výroby jinou hodnotu.

mask
Maska adresy zařízení. Defaultně přednastavena na 255.255.255.0. Je možné však vyžádat z výroby jinou hodnotu.

port
TCP port na kterém probíhá komunikace. Defaultně 3000.

max_connections
Počet paralelních spojení klientů se zařízením ETH2CAN. Povolené hodnoty 2..4. Menší počet spojení dovoluje zrychlit množství přenášených dat.

mac

MAC adresa zařízení. Defaultně 00-04-A3-00-00-00.

flash_period
Perioda ukládání dat do FLASH paměti v ms. Nastavitelný rozsah 200ms-20s.

app_start_delay
Čas v sekundách jak dlouho čeká zařízení po zapnutí na případný pocel ke změně firmware.

Paket FIRMWARE VERSION

Tímto paketem jsou je vyžadována verze firmware v zařízení interface ETH2CAN.

Požadavek klienta má tvar:

```
typedef struct _ETH_FIRMWARE_REQ {
     unsigned __int32    magic;
     unsigned char       packet_type;
     unsigned __int16    packet_size;
     unsigned char       id;
}    ETH_FIRMWARE_REQ;
```

Odpověď interface ETH2CAN má tvar:

```
typedef struct _ETH_FIRMWARE {
     unsigned __int32    magic;
     unsigned char       packet_type;
     unsigned __int16    packet_size;
     unsigned char       id;
     unsigned char[20]   fw_version_string;
}    ETH_FIRMWARE;
```

Položka obsahuje string s verzí firmware. Neobsahuje ukončovací 0 řetězce. Řetězec má tvar například CANLABsro-01.10. V režimu bootloaderu pak CANLABsro-01.10boot. Bootloader využívá jiné číslování než aplikace!

Paket STATUS

```
typedef struct _ETH_STATUS {
     unsigned __int32 magic;
     unsigned char packet_type;
     unsigned __int16 packet_size;
     unsigned char id;
} ETH_CAN_STATUS;
```

Tento paket je třeba zasílat cca každých 500ms z klientské aplikace do zařízení ETH2CAN. Tímto paketem je udržováno spojení. Pokud zařízení ETH2CAN nedetekuje v intervalu 5 sekund příchod tohoto paketu, spojení ze strany ETH2CAN zrušeno.

```
typedef struct _ETH_STATUS {
     unsigned __int32    magic;
     unsigned char       packet_type;
     unsigned __int16    packet_size;
     unsigned char       id;
     unsigned char       TEC;
     unsigned char       REC;
     unsigned __int16    RST;
     unsigned char       TXFIFO;
     unsigned char       RXFIFO;
     unsigned __int16    ETHTXFIFO;
     unsigned __int16    ETHRXFIFO;
} ETH_STATUS;
```

Paket je generován v intervalu cca 500ms ze zařízení ETH2CAN. Interval roste při větším zatížení, neboť generování tohoto paketu má nízkou prioritu. Klientská aplikace je tímto paketem informována o interních stavech interface ETH2CAN.

Paket VEHICLE_TYPE

Paket je určen k ověření jaký typ vozidla byl detekován.

Požadavek klienta má tvar:

```
typedef struct _ETH_VEHICLE_TYPE {
     unsigned __int32    magic;
     unsigned char       packet_type;
     unsigned __int16    packet_size;
     unsigned char       id;
}    ETH_VEHICLE_TYPE;
```

Odpověď interface ETH2CAN má tvar:

```
typedef struct _VEHICLE_TYPE {
     unsigned __int32    magic;
     unsigned char       packet_type;
     unsigned __int16    packet_size;
     unsigned char       id;
     unsigned char       vehicle_type;
}    ETH_VEHICLE_TYPE;
```

Vehicle type:

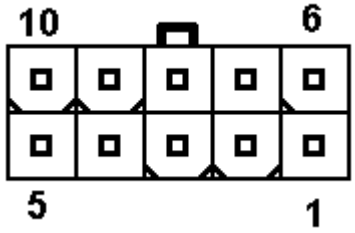
- 0 vozidlo dle SAE 1939
- 1 vozidlo SOR EURO 6

Připojení zařízení

Zařízení je uloženo v krabičce TOPTEC 102 firmy OKW. Zařízení má 2 konektory. Prvním je konektor RJ45, tedy klasický ethernetový konektor. Druhým konektorem je konektor MOLEX, který je určen při připojení napájení, vozidlové sběrnice CAN a dalších signálů.

Zařízení pracuje s rozsahem napájecího napětí 8-36V. Spotřeba zařízení v provozním stavu je 1.0W. V deaktivovaném stavu po odpojení signálu 15 je spotřeba rovna téměř nule. Signál 15 je aktivován cca od úrovně 1V.

Konektor MOLEX



Konektor na PCB.

Pin	Popis
1	Napájecí napětí 8-36V
2	GND
3	CAN H
4	GND
5	Digitální výstup, PLUS, rezerva
6	Signál 15 (startup-shutdown)
7	Vstup, dráhové pulzy
8	CAN L
9	Digitální vstup – rezerva
10	Digitální výstup, MINUS, rezerva

Funkce indikačních LED

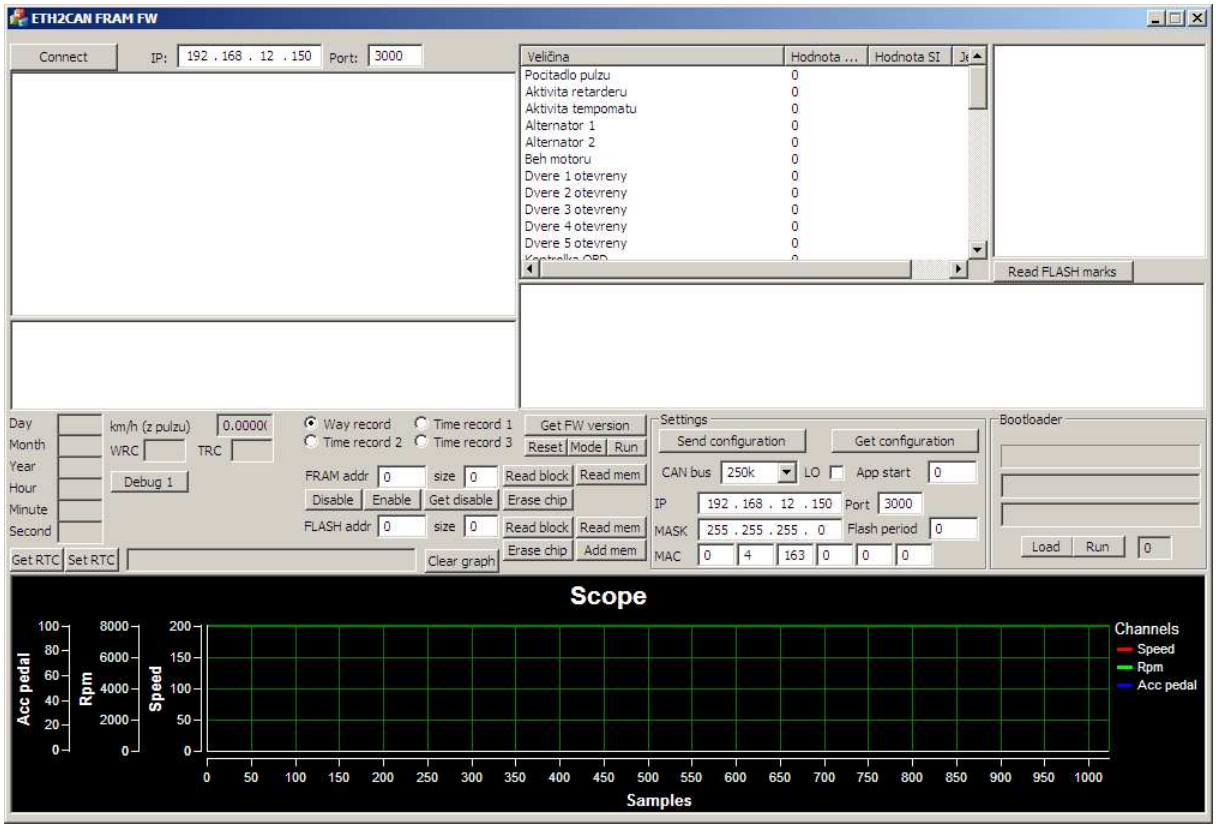
LED#	Barva	Popis
1	**GREEN**	Indikuje sepnutí signálem 15.
2	**YELLOW**	Mění stav pokud je ze zařízení odesílán paket s informacemi z vozidla (WAY a TIME RECORD).
3	**YELLOW**	Mění stav při příchodu paketu po rozhranní ethernet, je li hlavička paketu ve známém tvaru.
4	**YELLOW**	CAN RX, mění stav při příchodu CAN zprávy s informací o rychlosti vozidla (PGN 0xFEF1)
5	**RED**	Změna stavu indikuje sepnutí vstupu dráhových impulzů
5	**GREEN**	Indikuje příchod paketu (TCPIP paket, ping apod.)
6	**YELLOW**	Indikuje připojení ethernetového kabelu.

LED 2-5	Trvalý svit indikuje že zařízení je v režimu bootloaderu a čeká na spuštění aplikace nebo započetí změny firmware. Během změny firmware všechny LED blikají.

Zkratovací propojka **SW1** je určena pro aktivaci zakončovacího odporu 120 ohmu na CAN sběrnici. CAN sběrnice je vždy zakončena na obou stranách zakončovacími odpory 120 ohmu.
Zkratovací propojka **SW2** je určena k resetu nastavení. Po aktivaci s touto propojkou je nastavena defaultní IP adresa 192.168.12.150.
Zařízení obsahuje baterii pro zálohování hodin reálného času. Jedná se o typ BR1225. Minimální životnost baterie je 3 roky.

Výměna firmware

Výměna firmware je možná prostřednictvím rozhranní ETHERNET. Dodávaná testovací aplikace osahuje možnost načíst soubor nového firmware a zapsat jej do zařízení. Před započetím změny firmware je vhodné nastavit parametr **app_start_delay** na hodnotu 10 sekund. Následně provést změnu firmware a parametr nastavit zpět na hodnotu 1 sekunda.



Změny ve verzích firmware

1.00 boot

- první veřejná verze bootloadru

1.00

- první distribuční verze FW

1.50

- samostatná data TIMEDATA1, TIMEDATA2 a TIMEDATA3 nahrazeny jedním paketem TIMEDATA
- doplněna blokace zápisu do FRAM

1.60

- přidána TIME_RECORD_4. Tento záznam je odesílán na ETH a ukládán do FLASH. Není ukládán v FRAM.
- přidán PACKET_INPUTS a PACKET_OUTPUTS

2.00

- úprava pro vážení kloubového autobusu

3.00

- přidána extended timedata pro data ASC1 a ASC3

3.02

- přidán příkaz pro odemčení přístupu ke konfiguraci

4.00

- implementace změn ve vozidlech SOR EURO 6. Doplnění detekce typu vozidla – SOR EURO 6.