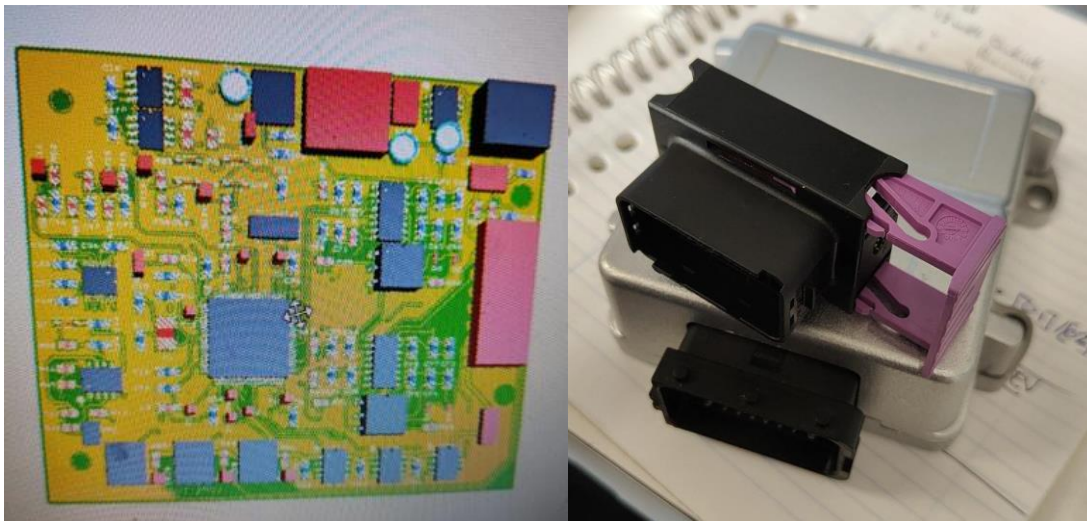# Programmers Manual

(with latest stm_xxx firmware loaded)

Battery Gateway / Blue BMS / ESS Firmware

(for board V3.1)

By : EV-Builder

Forum : OpenInverter.org

Version : v3.2 (31-10-2022)

Inhoud

History Notes

Change Log:

Change #	Ver.	Description
1	2.2	Added Mux to signal 0x603 to decrease bus load;
2	2.3	Added Cell Balance commands and status readout.
3	2.4	Added BMS status to CANID 0x600;
4	2.5	Added Module Count to CANID 0x600;
5	2.6	Changed/Fixed Cell Balance commands;
6	2.7	Changed strategy to upload balance config and added activate command;
7	2.8	Clarified module count during config and during discovery;
8	2.9	Added parameters & commands to change BMS Port en Canbus speed of EXT1
9	3.0	Needed more cell voltage space so moved the messages around; Check Interfaces section!
10	3.1	Added sections for separate firmware's behaviour
11		

Known Issues:

Issue #	HW Ver.	Description
1	2.0	By default only 3GP_INPUTS are connected to CPU.
2	2.0	PWM2 needs two CPU pins connected, shorted. (check for shorted pins)
3	Manual	Missing bit definition;
4	Manual	Parameter list not filled in yet;
5		
6		
7		

Foreword

This manual tries to be as complete as possible but please just mail and ask if certain wordings or explanations aren't clear enough.

The device can come with 3 types of firmware. It's important to untherstand the three types.

- BGTW   ; Battery Gateway; Often used within systems of integrators;
- BlueBMS   ; Vehicle BMS mode; The firmware will do typical BMS tasks on its own;
- ESSBMS   ; Energy Storage Systems; The firmware will communicate with hybrid inverters.

In the manual we use to indicate firmware appliance.

To indicate on which part the sections applies too.

The hardware variations are indicated like this:

Bootloader

The bootloader used is a CAN bootloader.

The CAN bootloader listens only on the **EXT2** interface @ 500K.

In order to use the CAN bootloader the matching GUI application is needed (see PEAK BMS Studio / PEAK Bootloader).

Select an interface, at the time of writing only the PEAK adapters and the BUSMUST adapter are supported.

Select a HEX file, upon succesfull reading some properties of the file will be shown;

After a reboot the device should pop-up in the list or if you know the device select it from the list with a double click.

Next reboot the device will receive the hex file, see the log tab for user friendly details and messages.

CAN ID **0x7DE** is used by the nodes and **0x7DD** is used by the bootloader application.

Identification of the Node by the GUI application is done by its unique chip ID. All devices will be kept in "wait_state" until a choice is made in the GUI application.

Configuration

The configuration consists of a hardware part, mainly on the PCB with jumpers and or optional population of components.

In the table below can be seen what would be added / sacrificed with each choice.
(actions might be needed like removal or placement of components and or closing / opening of solder jumpers).

HW Configurations HWv3.1:

HW Configuration Matrix CMC							
Default							
CanBus 1							
Canbus 2							
TPL 1 port							
3 Dig Inputs							
4 Dig Outputs							
Exclusive / OR Configuration options	+1 Dig Inputs	Analog 1 Input	Analog 2 Input	Analog 3 Input	Analog 4 Input	PWM GP-OUT1	PWM GP-OUT2
LIN Interface	X						
PWM1_Out		X					
PWM2_Out			X				
TPL 2 port				X	X		
RS232 Interface				X	X		
PWM1 Driver						X	
PWM2 Driver							X

[Config. Table: When an X is read in the junction it means that a choice has to be made between those 2 functions.]
(Contact us for latest updates of this table).

SW Configuration & Commands

Configuration of the device is done over CANBus. The settings only have to be sended <u>once</u> per reset cycle (no saving of settings currently supported). The device will listen on both CANBus ports for the configuration.

CAN ID **0x598** is the ID used for sending over the settings on the **EXT2 or EXT1** interface.

All bytes are unsigned (dec. 0-255);

Description	Default	Byte0(CMD)	Setting parameters	Remarks
PORT (Enable/Disable)	Disabled	0x1	Byte1=Port#, Byte2=0/1	Please delay 1000ms between switching. (modules need to time-out)
Nr of Modules	1	0x2	Byte1=PORT#; Byte2=Number of modules	Byte1 = 1 or 2; Byte2 = 1-15;
Change BMSport and EXTSpeed		0x5	Byte1=BMSPort;Byte2=EXT1Speed;	Byte1=1 for EXT2 (default) or 2 for EXT; Byte2=0 for 250K, 1 for 500K and 2 for 800K;
Write Sink Outputs	Disabled	0x10	Byte1=3Bits LSB are on / off	
Enable/Disable Global Balancing for module(s)		0x20	Byte1=Port#, Byte2=Enable/Disable Byte3=MOD#	Byte1 = 1 or 2; Byte2 = 0 or 1; Byte3 = 0 (all); 1 to 15;
Enable 2Min. Timer Balancing for a cell(s); (re-write means restart of timer)		0x21	Byte1=Port#, Byte2=CELL MASK Byte3=MOD# byte4=Activate Config	Byte1 = 1 or 2; Byte2 = 0 all off; 1 to 63; Byte3 = 0 (all mod.); 1 to 15; Byte4 = 0 or 1

Cell Balancing

Cell balancing can be activated on cell level. The bleed resistors on the internal PCB will be enabled by command. This will bleed energy out of certain cells. In that way we try to get as many cells as possible at a high charge level.

Step 1) Wait for BMS Status == Running;
Step 2) Enable / Disable Global Balancing;
Step 3) Wait for BMS Status == Running;
Step 4) Upload Cell balance config per module;
Step 5) Set Activate bit in last Cell level balancing config

Now the whole config will be distributed to the pack at once.
In this way we interrupt the measurements the least possible and we minimize the risk of tail trailing our own changes.

MOD#: 0= All modules; 1-15 --> one module command;
CELL#: 0= All Cells off; 1 = cell ON; 3= 1 & 2 On etc. (bit pos. 1 = Cell 1 etc.)

Main Connections

(Default HW configuration)

TPL	TPL	MAX

TPL (<= HW2.0)

Signal	Pin	No.
+12Vin	A1	1
GND	B1	2
GND	C1	3
CCM1_P	A2	4
CCM0_P	B2	5
LIN_BUS	C2	6
CCM1_N	A3	7
CCM0_N	B3	8
CAN_EXT2H	C3	9
CAN_EXT2L	A4	10
CAN_EXTH	B4	11
CAN_EXTL	C4	12
GP_OUT1	A5	13
GP_OUT2	B5	14
GP_INP_2	C5	15
GP_INP_0	A6	16
GP_INP_1	B6	17
PWM1_OUT	C6	18
PWM3_OUT	A7	19
GP_OUT3	B7	20
PWM2_OUT	C7	21
+12Vin_PWM	A8	22
RS232_Rx	B8	23
RS232_Tx	C8	24

TPL (HW3.x)

Signal	Pin	No.
+12Vin	A1	1
GND	B1	2
GND	C1	3
CCM1_P	A2	4
CCM0_P	B2	5
LIN_BUS	C2	6
CCM1_N	A3	7
CCM0_N	B3	8
CAN_EXT2H	C3	9
CAN_EXT2L	A4	10
CAN_EXTH	B4	11
CAN_EXTL	C4	12
GP_OUT1	A5	13
GP_OUT2	B5	14
GP_INP_2	C5	15
GP_INP_0	A6	16
GP_INP_1	B6	17
PWM1_OUT	C6	18
RS232_Rx	A7	19
RS232_Tx	B7	20
PWM2_OUT	C7	21
+12Vin_PWM	A8	22
GP_OUT4	B8	23
GP_OUT3	C8	24

MAX (HW3.x)

Signal	Pin	No.
+12Vin	A1	1
GND	B1	2
GND	C1	3
CCMRX_P	A2	4
CCMTX_P	B2	5
LIN_BUS	C2	6
CCMRX_N	A3	7
CCMTX_N	B3	8
CAN_EXT2H	C3	9
CAN_EXT2L	A4	10
CAN_EXTH	B4	11
CAN_EXTL	C4	12
GP_OUT1	A5	13
GP_OUT2	B5	14
GP_INP_2	C5	15
GP_INP_0	A6	16
GP_INP_1	B6	17
PWM1_OUT	C6	18
RS232_Rx	A7	19
RS232_Tx	B7	20
PWM2_OUT	C7	21
+12Vin_PWM	A8	22
GP_OUT4	B8	23
GP_OUT3	C8	24

Connector PN	: **211PC249S0018**
BIG PIN	:
SMALL PIN	:

<= HW2.0   **| HW3.x**   **| HW3.x**

Notes:

The +12Vin_PWM is fused on the board with an SMD fuse (5A Chip Fuse 19mΩ Slow Blow 2410 Surface Mount Fuses ROHS);

All outputs are diode (relais) protected so **connect** the +12Vin_PWM pin to coil supply line.

- The +12Vin is diode polarity protected;
- The GP_OUTx are sinking outputs. A pull-up can be activated (HW2.0 only) with a jumper on the print (default 4K7), power comes from fused PWM main connection;
- CCMxP / CCMxN the string connections from the MC33664 to the Battery daisy chain. 1:1 Positive to Positive and Negative to Negative.

Digital Outputs
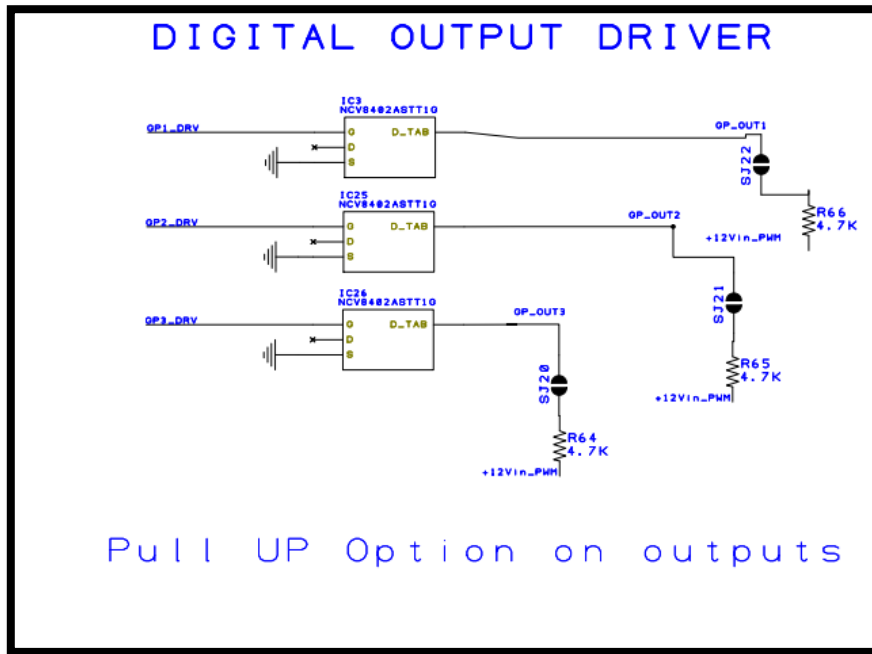
Note: HW2.0 | Outputs are current sinking (switching low side), as of HW V3.0 pull-ups are replaced with diodes.
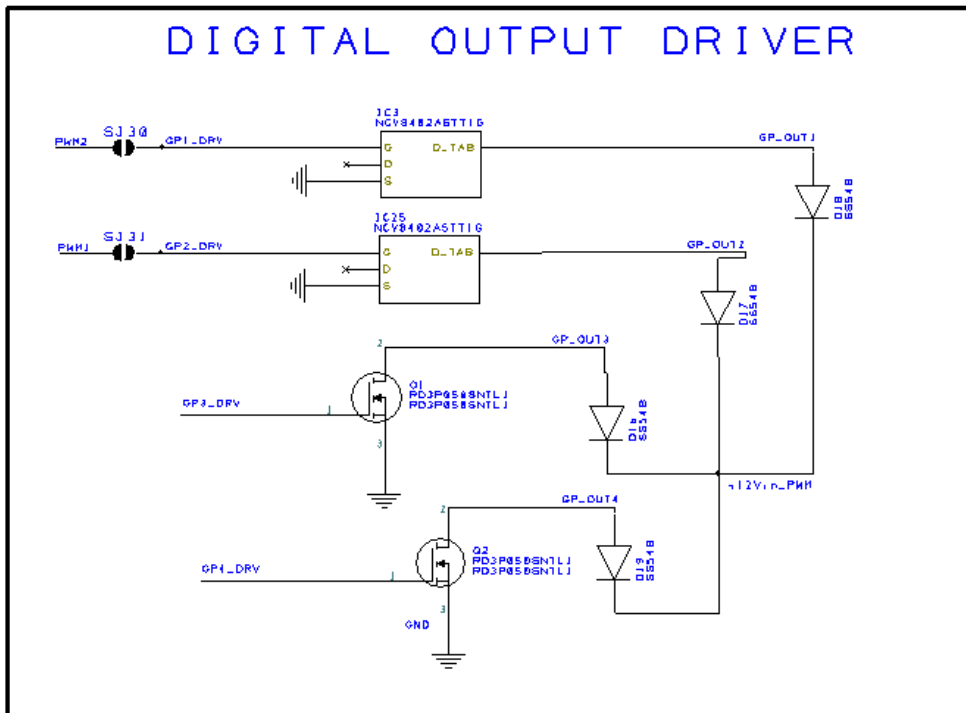
Note: HW3.0 | Outputs are current sinking (switching low side) with fly-back diodes onboard, connect +12Vin_PWM to coil supply;

Attention! : Depending on wire lengths and thickness this might <u>not</u> be sufficient, please make sure the output stage is protected.

Digital Inputs

3 GP DIGITAL INPUTS 12V

SJ27 is Shared with LINBUS PIN on CONNECTOR
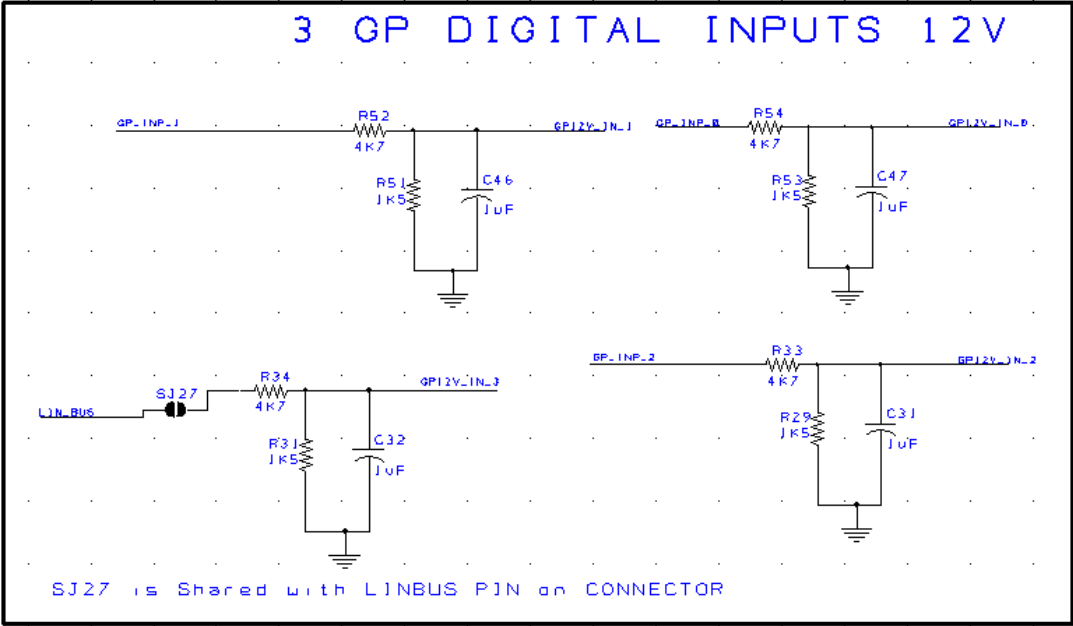
Note: Due to density on the PCB a simple filter and voltage divider forms the input circuitry. +14.4VDC is maximum! (Change resistors R52/R54/R34/R33, 0508 package, if higher voltage is desired). PCB V2.0 design, by default only 3GP_INPUTS are connected to CPU.

Connecting a battery module

Porsche TayCan
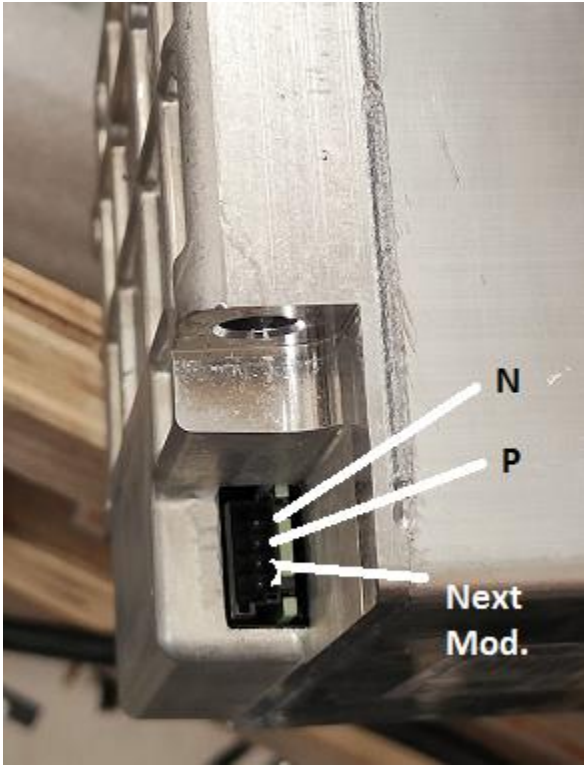
The CCMx_N pin from the controller goes to the TOP pin in this picture.
The CCMx_P pin goes to the second pin (P pin). The next module is connected to pin 3 and pin 4.

At the time of writing the TayCan battery supports:

- Module voltage;
- Cell voltages;
- 2 Temperature sensors;
- 1 Chip temperature;
- Cell balancing;
- Temperature latch bits;
- Voltage latch bits;
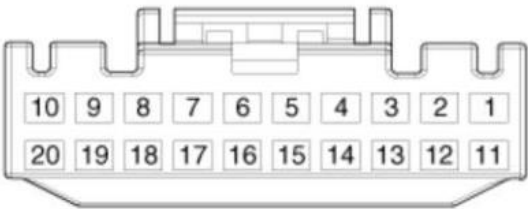
B Connector [B01-1B] (20pin)

Connector B has the 4 wires of the internal dataBUS, connection to the first module.

Connect as follow:

Terminal on BMS B	Signal Name	On CMC MAX
1 (Bl)	Module Nr1, TXNL	RxN (A3)
2 (Rd)	Module Nr1, TXPL	RxP (A2)
11 (BlWh)	Module Nr1, RXNL	TxN (B3)
12 (RdWh)	Module Nr1, RXPL	TxP (B2)

(Note: Hyundai nomenclature is used)

Each CMC (aka Device) has 4 incomeing connections and 4 outgoing connections to connect to the next modules incomeing port. In that way a ring is formed. It's important that the last module incorporates the loopback connection.

So please make sure its present in your loom.
The loopback connection makes sure the chain forms a ring:

Signal Name	On Cell Module
Last Module Nrx, TXUN	Last Module Nrx, RXUN
Last Module Nrx, TXUP	Last Module Nrx, RXUP

(Note: Maxim nomenclature is used)

Interfaces

CAN BUS

At the time of writing the CANBus data is present, by default, on the **EXT2** output (A4 and C3 pins). It's also distributed if the GUI is connected.

On initialization of the device, after the command is received, the battery string is explored for the configured devices. If the configured amount of devices is found and configured with a default internal configuration the internal BMS state of the bgtw module will change to running. In the running state the values of the battery modules are cyclically (50mS) send over the CanBus @500Kbs.

CAN ID	Direction	Description
0x598	IN	Listening for configuration
0x599	OUT	IO status; Input states, Output states etc.
0x600	OUT	This message sends the Battery module and Port number; It's like a header in normal comm.; Contains Module total voltage; Temperature etc.
0x601	OUT	Cell voltages
0x602	OUT	Cell voltages
0x603	OUT	Cell voltages
0x604	OUT	Muxed (1-5): Analog DAC voltages / Coulomb counter / Analog To NTC Celsius
0x605	OUT	Threshold Bits

With parameters (CMD:0x5) the BMSPort can be moved and a different speed can be set for the **EXT** canbus port.

Output Data description

The header file is available for implementation on the receiving end of the solution.

```c
struct Canx599{
    uint8_t Inputs;
    uint8_t Outputs;
    uint16_t uint16_tPWM1;
    uint16_t uint16_tPWM2;
    uint16_t uint16_tPWM3;
};

* typedef enum {
    BMS_Unknown     = 0,                                    //!< used by GUI
    BMS_Init        = 1,                                    //!< init phase is assigning CID
    BMS_Config      = 2,                                    //!< config phase applies initial loading of registers
    BMS_Running     = 3,                                    //!< bms is running
    BMS_Sleeping    = 4,                                    //!< bms is sleeping
    BMS_Error       = 5,                                    //!< error phase
    BMS_Idle        = 6,                                    //!< BMS_Idle
    BMS_Off         = 7,                                    //!< BMS_Off
    BMS_ConfigMainCB    = 8,
    BMS_ConfigCellCB    = 9
TYPE_BMS_STATUS; */

struct Canx600{
    uint8_t  CID;                //block # in order of daisy chain & ( NUMBER OF MODULES FOUND << 4 )
    uint8_t  BMSNum;             //#of string connected to the device & ( TYPE_BMS_STATUS << 4 );
    uint16_t uint16_tStackVoltage;
    uint16_t uint16_tTChip;      //miliKelvin / bit
    uint8_t  uint16_t2;
};

struct Canx601{
    uint16_t uint16_tCell1Voltage;
    uint16_t uint16_tCell2Voltage;
    uint16_t uint16_tCell3Voltage;
    uint16_t uint16_tCell4Voltage;
};

struct Canx602{
    uint16_t uint16_tCell5Voltage;
    uint16_t uint16_tCell6Voltage;
    int32_t int32_tCurrent;     //6ma / bit
};
```

Note:

When the device receives the module count config from the user control unit, it responses back with that number of modules (Message 0x600; byte 0; mask 0x0F);
After the string is activated the config value will be overwritten with the actual number of modules found during discovery and subsequently programming of the modules.

This serves two purposes:

1) **User can check if correct amount is configured before activation of the string (starting discovery);**
2) **User can check runtime if correct amount of modules has been identified on the bus (if the status is running, the value is valid);**

```c
struct Canx603_1{
    uint8_t  datamux; //=0x1;
    uint8_t spare;
    uint16_t uint16_tAN1Voltage;
    uint16_t uint16_tAN2Voltage;
    uint16_t uint16_tAN3Voltage;
};

struct Canx603_2{
    uint8_t  datamux; //=0x2;
    uint8_t spare;
    uint16_t uint16_tAN4Voltage;
    uint16_t uint16_tAN5Voltage;
    uint16_t uint16_tAN6Voltage;
};

struct Canx603_3{
    uint8_t  datamux; //=0x3;
    uint8_t spare;
    uint16_t uint16_tAN7Voltage;
    uint16_t uint16_tspare;
    uint16_t uint16_tspare2;
};

struct Canx603_4{
    uint8_t  datamux; //=0x4;
    uint8_t  uint8_CBStatus; //the CellBalanceSwitchStatus is only 6BITS
    uint16_t uint16_tCCSamples;
    int32_t  int32_tCoulombCounter;
};

struct Canx603_5{
    uint8_t  datamux; //=0x5;
    uint8_t uint8_tAN1Voltage;
    uint8_t uint8_tAN2Voltage;
    uint8_t uint8_tAN3Voltage;
    uint8_t uint8_tAN4Voltage;
    uint8_t uint8_tAN5Voltage;
    uint8_t uint8_tAN6Voltage;
    uint8_t uint8_tAN7Voltage;
};
```

```c
struct Canx604{
    uint16_t uint16_tSpare;
    uint16_t uint16_tSpare2;
    uint8_t uint8_tThresholdUVBITS; //under voltage
    uint8_t uint8_tThresholdOVBITS; //over voltage
    uint8_t uint8_tThresholdUTBITS; //under Temperature
    uint8_t uint8_tThresholdOTBITS; //over Temperature
};
```

Calculations

[0x600 t/m 0x603]

```
if (msg.ID == 0x600)
{
    fieldnr = Convert.ToInt32(msg.DATA[0]);

    dgvBMS.Rows[PACKVOLTS].Cells[fieldnr + 1].Value = FormatS((((msg.DATA[2] << 8) + msg.DATA[3])  * 24.4414) /10000);
    dgvBMS.Rows[TCHIP].Cells[fieldnr + 1].Value = FormatS(((((msg.DATA[4] << 8) + msg.DATA[5]) * 32) /1000.0) -273.0);

}
else if (msg.ID == 0x601)
{
    dgvBMS.Rows[CELL1].Cells[fieldnr + 1].Value = FormatS(( ( (msg.DATA[0] << 8) + msg.DATA[1]) * 1.5259) / 10000);
    dgvBMS.Rows[CELL2].Cells[fieldnr + 1].Value = FormatS(( ((msg.DATA[2] << 8) + msg.DATA[3]) * 1.5259) / 10000);
    dgvBMS.Rows[CELL3].Cells[fieldnr + 1].Value = FormatS(( ((msg.DATA[4] << 8) + msg.DATA[5]) * 1.5259) / 10000);
    dgvBMS.Rows[CELL4].Cells[fieldnr + 1].Value = FormatS(( ((msg.DATA[6] << 8) + msg.DATA[7]) * 1.5259) / 10000);
}
else if (msg.ID == 0x602) {
    dgvBMS.Rows[CELL5].Cells[fieldnr + 1].Value = FormatS(( ((msg.DATA[0] << 8) + msg.DATA[1]) * 1.5259) / 10000);
    dgvBMS.Rows[CELL6].Cells[fieldnr + 1].Value = FormatS(( ((msg.DATA[2] << 8) + msg.DATA[3]) * 1.5259) / 10000);

    Int32 ti32 = 0;
    byte[] barr;
    barr = msg.DATA;

    ti32 = BitConverter.ToInt32(barr, 4);

    dgvBMS.Rows[CURRENT].Cells[fieldnr + 1].Value = FormatS((ti32 * 6) / 1000);

}
```

```
else if (msg.ID == 0x603)
{
    int mux = msg.DATA[0];


    //int temp = ((msg.DATA[0] << 8) + msg.DATA[1]);
    //dgvBMS.Rows[ANCELL1].Cells[fieldnr + 1].Value = FormatS((temp * 1.5259) / 10000);

    int temp = ((msg.DATA[2] << 8) + msg.DATA[3]);

    if (mux == 1)
    {
        dgvBMS.Rows[ANCELL1].Cells[fieldnr + 1].Value = FormatS( (temp * 1.5259) / 10000);

        temp = ((msg.DATA[4] << 8) + msg.DATA[5]);
        dgvBMS.Rows[ANCELL2].Cells[fieldnr + 1].Value = FormatS( (temp * 1.5259) / 10000);

        temp = ((msg.DATA[6] << 8) + msg.DATA[7]);
        dgvBMS.Rows[ANCELL3].Cells[fieldnr + 1].Value = FormatS( (temp * 1.5259) / 10000);

    } else if (mux == 2)
    {
        dgvBMS.Rows[ANCELL4].Cells[fieldnr + 1].Value = FormatS( (temp * 1.5259) / 10000);

        temp = ((msg.DATA[4] << 8) + msg.DATA[5]);
        dgvBMS.Rows[ANCELL5].Cells[fieldnr + 1].Value = FormatS( (temp * 1.5259) / 10000);

        temp = ((msg.DATA[6] << 8) + msg.DATA[7]);
        dgvBMS.Rows[ANCELL6].Cells[fieldnr + 1].Value = FormatS( (temp * 1.5259) / 10000);

    } else if (mux == 3)
    {
        temp = ((msg.DATA[2] << 8) + msg.DATA[3]);
        dgvBMS.Rows[CCSamples].Cells[fieldnr + 1].Value = temp.ToString();

        Int32 ti32 = 0;
        byte[] barr;
        barr = msg.DATA;

        ti32 = BitConverter.ToInt32(barr, 4);

        dgvBMS.Rows[CCounter].Cells[fieldnr + 1].Value = ti32;

    }
```

18

```
} else if (mux == 4)
{
    UInt16 tuin16 = Convert.ToUInt16((msg.DATA[3] << 8) + msg.DATA[2]);
    dgvBMS.Rows[CCSamples].Cells[fieldnr + 1].Value = tuin16.ToString();


    Int32 ti32 = 0;
    byte[] barr= new byte[8];
    barr = msg.DATA;



    ti32 = BitConverter.ToInt32(barr, 4);

    dgvBMS.Rows[CCounter].Cells[fieldnr + 1].Value = ti32;

    temp = (msg.DATA[1]); // + msg.DATA[4]);
    dgvBMS.Rows[CBStatus].Cells[fieldnr + 1].Value = Convert.ToString(temp, 2);

} else if (mux == 5)
{
    temp = (msg.DATA[1]);
    dgvBMS.Rows[ANTCELL1].Cells[fieldnr + 1].Value = FormatS((temp));// * 1.5259) / 10000);
    temp = (msg.DATA[2]);
    dgvBMS.Rows[ANTCELL2].Cells[fieldnr + 1].Value = FormatS((temp));// * 1.5259) / 10000);
    temp = (msg.DATA[3]);
    dgvBMS.Rows[ANTCELL3].Cells[fieldnr + 1].Value = FormatS((temp));// * 1.5259) / 10000);
    temp = (msg.DATA[4]);
    dgvBMS.Rows[ANTCELL4].Cells[fieldnr + 1].Value = FormatS((temp));// * 1.5259) / 10000);
    temp = (msg.DATA[5]);
    dgvBMS.Rows[ANTCELL5].Cells[fieldnr + 1].Value = FormatS((temp));// * 1.5259) / 10000);
    temp = (msg.DATA[6]);
    dgvBMS.Rows[ANTCELL6].Cells[fieldnr + 1].Value = FormatS((temp));// * 1.5259) / 10000);
    temp = (msg.DATA[7]);
    dgvBMS.Rows[ANTCELL7].Cells[fieldnr + 1].Value = FormatS((temp));// * 1.5259) / 10000);

}
```

Scaling

Data Parameter	Scale	
PACKVOLTS	2.44148	mV/LSB
CELLVOLTS	152.58789	µV/LSB
TCHIP	0.032	K/LSB
CURRENT	0.6	µV/LSB
CELL OV/UV	19.53125	mV/LSB
AN OT/UT	4.8828125	mV/LSB

Blue BMS / ESS BMS

Additional Connections

Pin	Function	Notes
GPOUT3	Main batterypack Coil, switch negative side of coil;	+12VDC 5Amp max. economized coils only;
GPOUT4	Pre-Charge Coil, switch negative side of coil;	+12VDC 5Amp max. economized coils only;
GPOUT2	Free Function 1	+12VDC 3Amp max. / current sinking;
GPOUT1	Free Function 2	+12VDC 3Amp max. / current sinking;

ESS BMS Parameters / Configuration

In the GUI all present parameters can be loaded from the device. With the button [**GET Settings**].