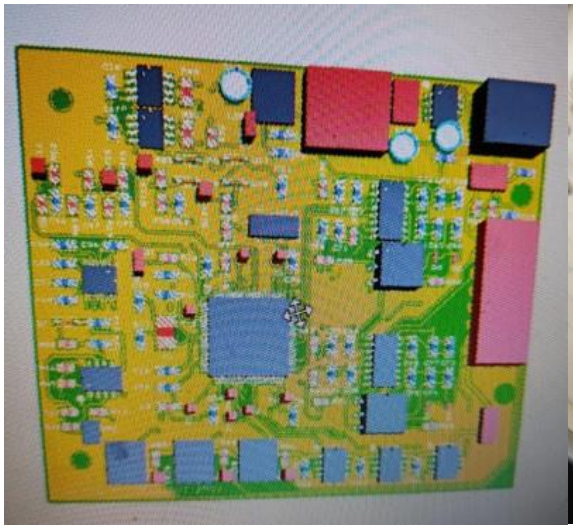
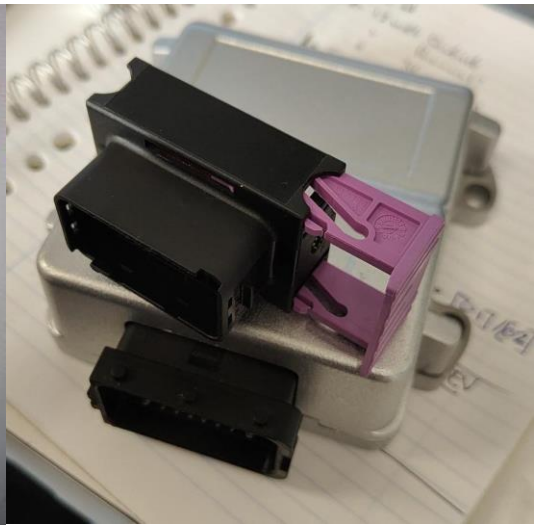
Programmers Manual

(with latest stm_bgtw firmware loaded)

Battery Gateway

(for board V2.0)





By : **EV-Builder**

Forum : **OpenInverter.org**

Version : **v2.6 (6-04-2022)**

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		
7		

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		
5		
6		
7		

Bootloader

The bootloader used is a CAN bootloader.

The CAN bootloader listens on the **EXT2** interface.

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 are supported.

Select a HEX file, some properties of the file should 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 application.

Configuration & Commands

Configuration of the device is done over CANBus. The settings only have to be sended once 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** interface.

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

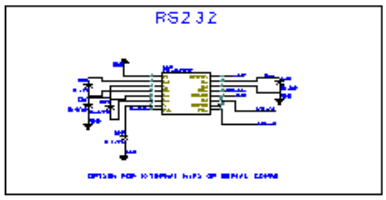
Description	Default	Byte0(CMD)	Setting parameters	Remarks
PORT (Enable/Disable)	Disabled	0x1	Byte1=Port#, Byte2=0/1	Please delay 1000ms between switching. (modules need to time-out)
Nr of Modules	1	0x2	Byte1=PORT#; Byte2=Number of modules	Byte1 = 1 or 2; Byte2 = 1-15;
Write Sink Outputs	Disabled	0x10	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#	Byte1 = 1 or 2; Byte2 = 0 all off; 1 to 63; Byte3 = 0 (all modules); 1 to 15;

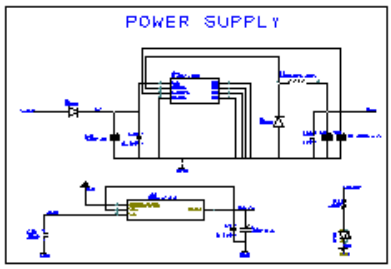
Cell Balancing

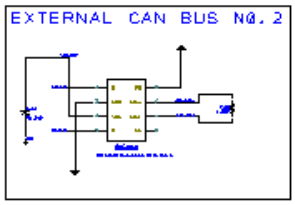
Step 1) Enable / Disable Global Balancing;
Step 2) Turn on Cell level balancing

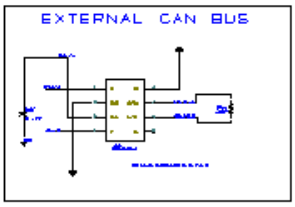
MOD#: 0= All modules; 1-15 --> one module command;
CELL#: 0= All Cells off; 1 = cell ON; 3= 1 & 2 On etc. (bit pos. 1 = Cell 1 etc.)

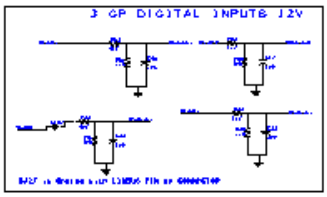
Schematic / Electrical

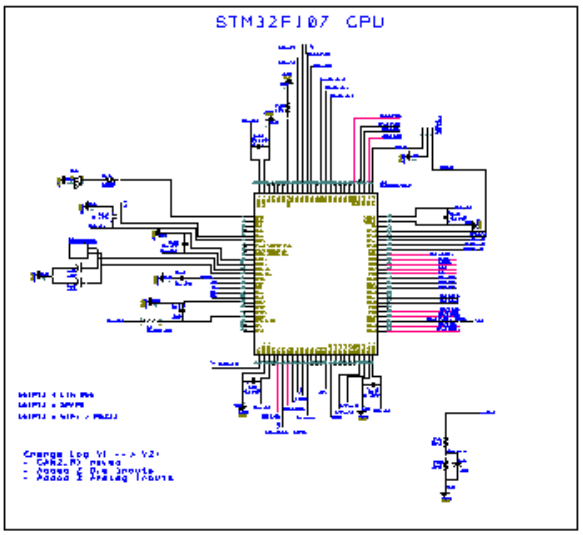


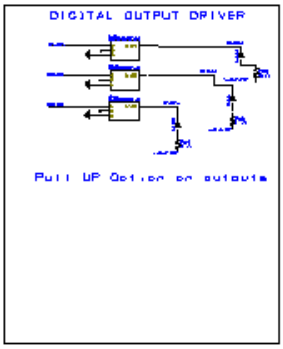


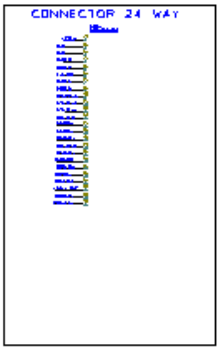


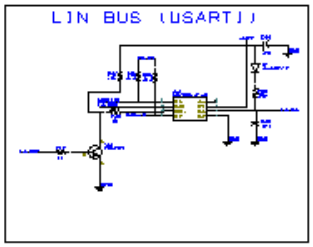


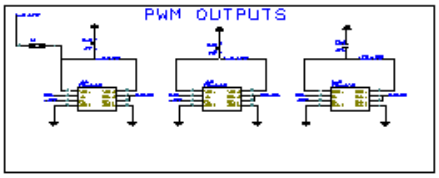


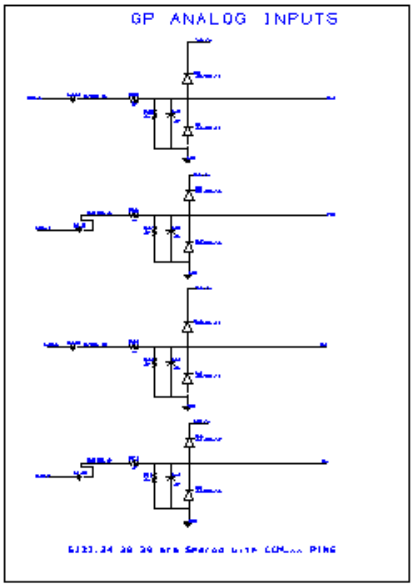


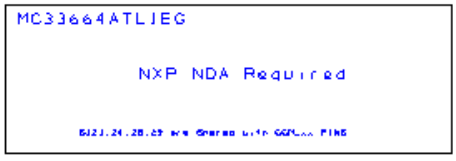


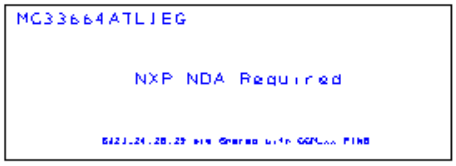












Main Connector

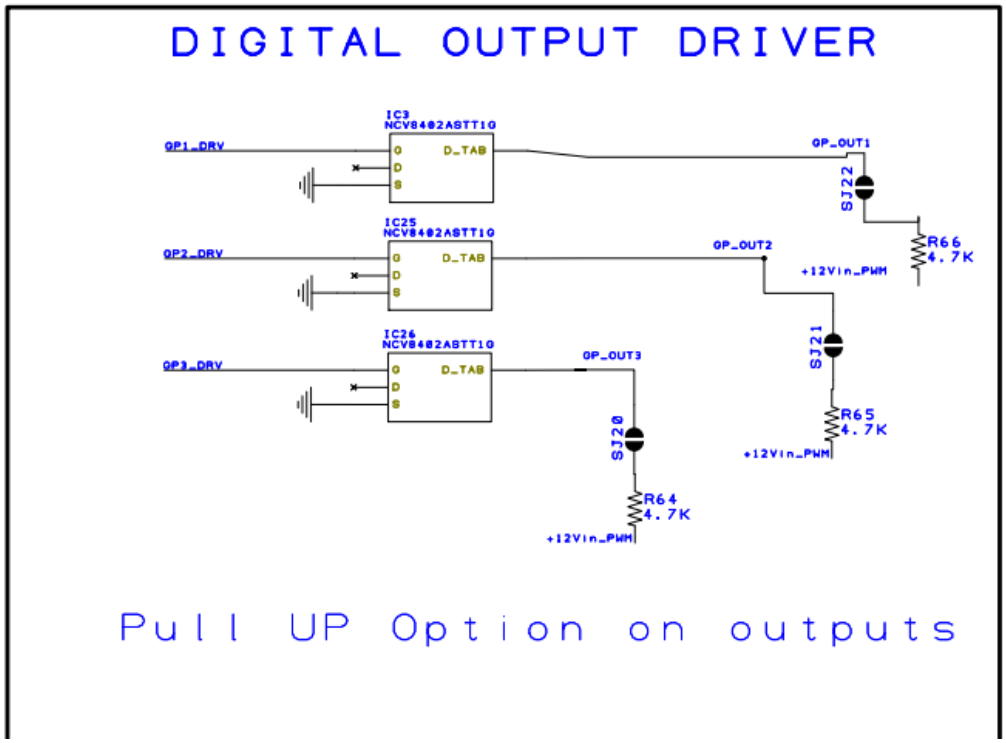
+12Vin	A1	1
GND	B1	2
GND	C1	3
CCM1_P	A2	4
CCM0_P	B2	5
LIN_BUS	C2	6
CCM1_N	A3	7
CCM0_N	B3	8
CAN_EXT2H	C3	9
CAN_EXT2L	A4	10
CAN_EXTH	B4	11
CAN_EXTL	C4	12
GP_OUT1	A5	13
GP_OUT2	B5	14
GP_INP_2	C5	15
GP_INP_0	A6	16
GP_INP_1	B6	17
PWM1_OUT	C6	18
PWM3_OUT	A7	19
GP_OUT3	B7	20
PWM2_OUT	C7	21
+12Vin_PWM	A8	22
RS232_Rx	B8	23
RS232_Tx	C8	24

Notes:

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

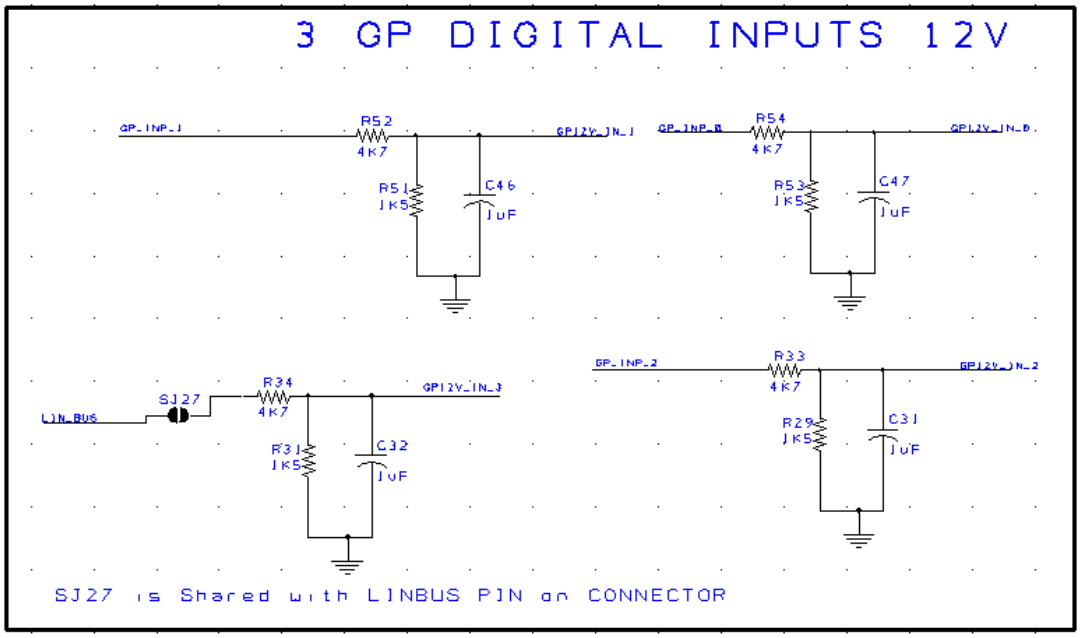
- The +12Vin is diode polarity protected;
- The GP_OUTx are sinking outputs. A pull-up can be activated with a jumper on the print (default 4K7), power comes from fused PWM main connection;
- CCMxP / CCMxN the string connections from the MC33664 to the Battery daisy chain. 1:1 Positive to Positive and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

Digital Outputs



Note: Outputs are current sinking (switching low side)

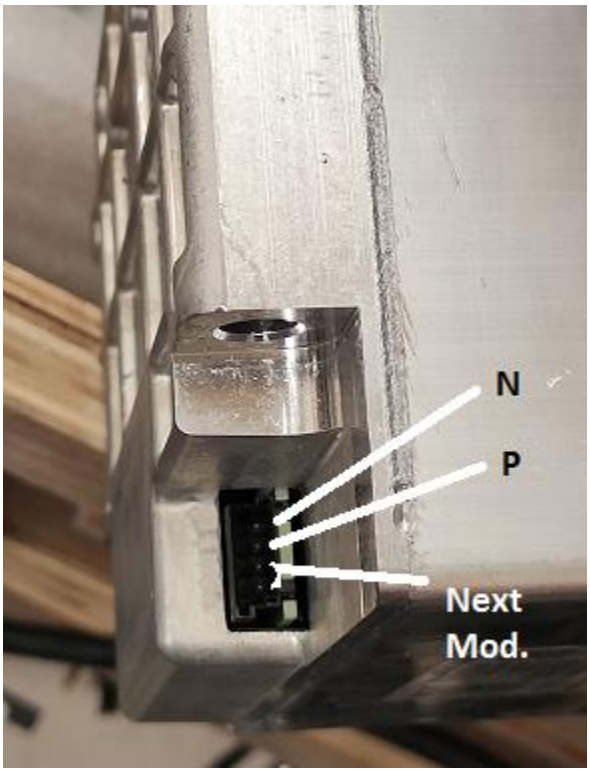
Digital Inputs



Note: Due to density on the PCB a simple filter and voltage divider forms the input circuitry. +14.4VDC is maximum! (Change resistors, 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 bottom (P pin).

Interfaces

CAN BUS

At the time of writing the CANBus data is present only on the **EXT2** output (A4 and C3 pins).

On initialization of the device the battery string is searched 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	Muxed (1-5): Analog DAC voltages / Coulomb counter / Analog To NTC Celsius
0x604	OUT	Threshold Bits

Output Data description

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

```
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;
    uint32_t uint32_tCurrent;    //6ma / bit
};
```

```
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;
 };

truct 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;
};
```

```
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 0x602]

```
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);

}
```

[0x603]

```
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;

    }
```

```
} 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