Volvo Motronic M4.3

A detailed description for Motronic 4.3 ECUs used in Volvo 850 T5(R)
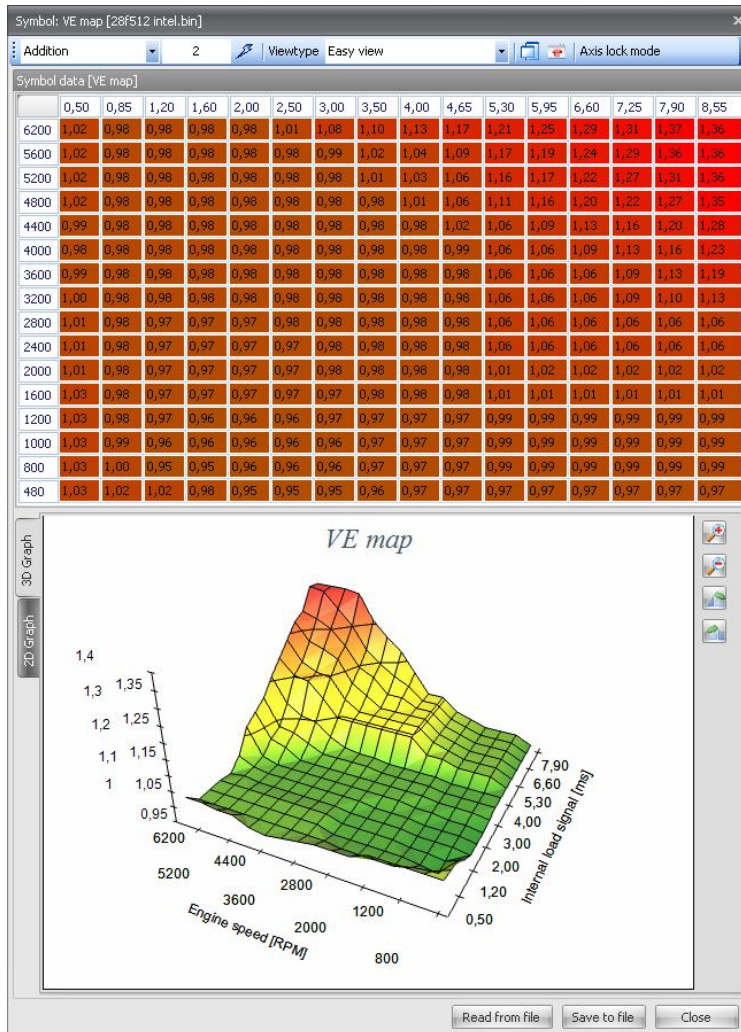
Table of content

Introduction

This document describes the Motronic M4.3 ECU in detail. It will first describe the hardware and proceed with a even more detailed description of the software that is running in the ECU so that we can learn how to tweak and tune the ECU to match the hardware – altered or not – that is on the car better.

Special thanks for getting all this together go out to rkam, T5_Germany, Steve Hayes and others on ecurproject.com, trionictuning.com and volvospeed.com.

Hardware

Overview of the board

The ECU contains a tri-layer printed ciruit board (PCB) which holds a lot of SMD components. The main component are – logically: Main CPU, Flash program storage, SRAM memory (working memory) and a lot of input/output (I/O).
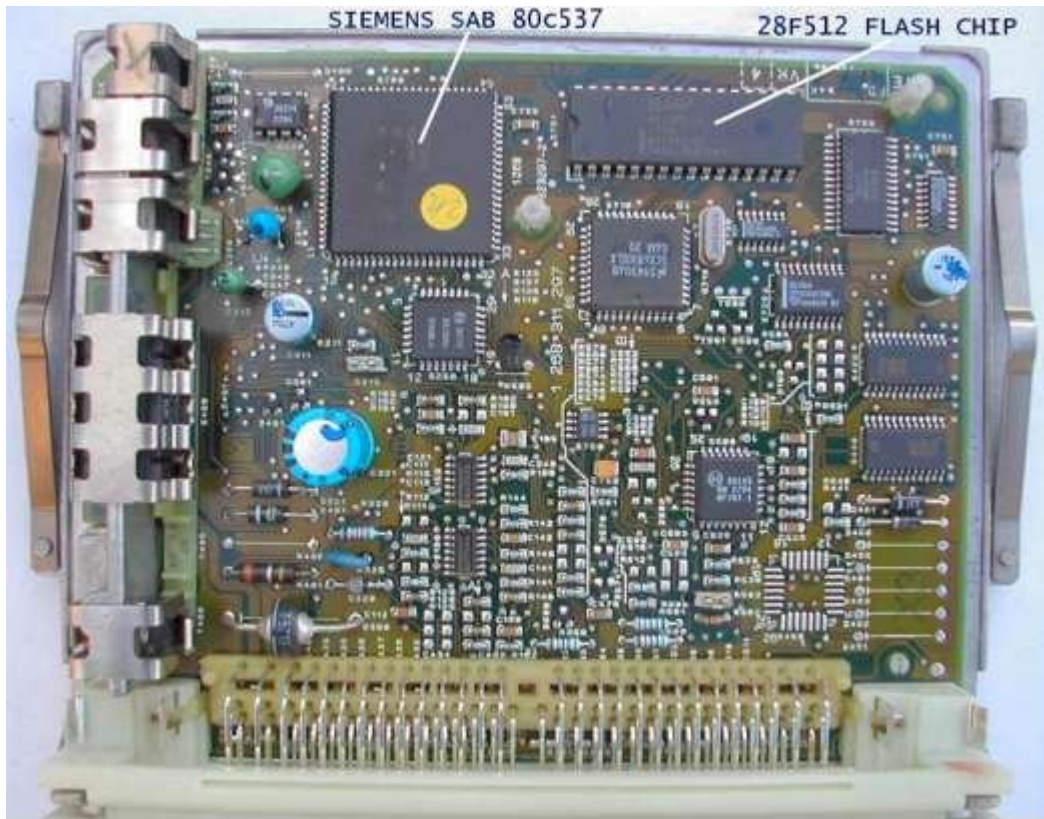
Main CPU: Siemens SAB 80C517

- ✓ 16 Mhz operating frequency
- ✓ 256 x 8 on-chip RAM
- ✓ 8k x 8 on-chip ROM for bootloader
- ✓ 64 Kbyte external data and program memory addressing
- ✓ 4 x 16-bit timer/counters
- ✓ Powerful 16-bit compare/capture unit with 21 x PWM outputs
- ✓ Versatile "fail-safe" provisions
- ✓ 8-bit A/D converter with 12 multiplexed inputs
- ✓ Two full duplex serial interfaces
- ✓ Nine ports: 56 I/O lines, 12 input lines

Datasheet documents

http://trionic.mobixs.eu/Motronic/M4.3/80c535.pdf

http://trionic.mobixs.eu/Motronic/M4.3/80c517um.pdf

SIEMENS SAB 80c537
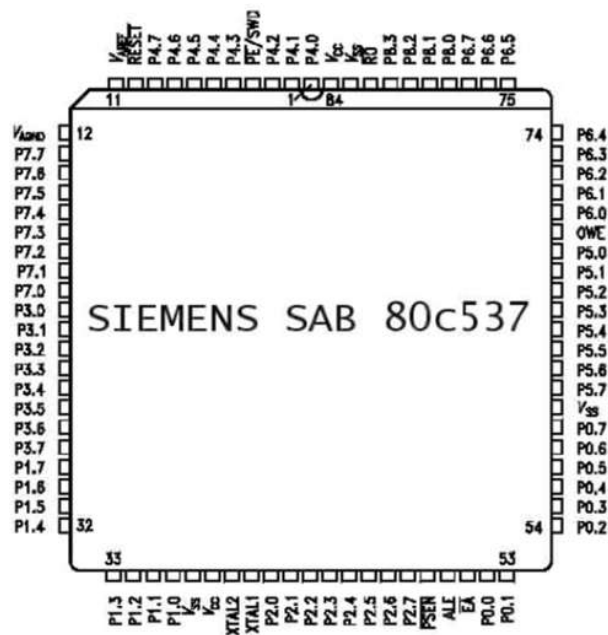
Flash eprom: AP28F512

Please note that very early M4.3 units has EPROMs in stead of flash chips. The correct number for these are 27C256. The early ECUs that have EPROMs are labelled "TOMAS".

```
         V_PP  ⊏ 1        32 ⊐ V_CC
          NC  ⊏ 2        31 ⊐ WE#
         A_15 ⊏ 3        30 ⊐ NC
         A_12 ⊏ 4        29 ⊐ A_14
          A_7 ⊏ 5        28 ⊐ A_13
          A_6 ⊏ 6        27 ⊐ A_8
          A_5 ⊏ 7  P28F512  26 ⊐ A_9
                   32 – PIN
          A_4 ⊏ 8    PDIP   25 ⊐ A_11
          A_3 ⊏ 9  0.62" x 1.64"  24 ⊐ OE#
                   TOP VIEW
          A_2 ⊏ 10       23 ⊐ A_10
          A_1 ⊏ 11       22 ⊐ CE#
          A_0 ⊏ 12       21 ⊐ DQ_7
         DQ_0 ⊏ 13       20 ⊐ DQ_6
         DQ_1 ⊏ 14       19 ⊐ DQ_5
         DQ_2 ⊏ 15       18 ⊐ DQ_4
         V_SS ⊏ 16       17 ⊐ DQ_3
```

ECU Pinout

```
                    A                                              B
  43 42 41 40 39 38 37 36 35 34 33 32 31 30    1  2  3  4  5  6  7  8  9 10 11 12 13 14
  29 28 27 26 25 24 23 22 21 20 19 18 17 16 15  15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
     14 13 12 11 10  9  8  7  6  5  4  3  2  1   30 31 32 33 34 35 36 37 38 39 40 41 42 43
```

Pin number	Color	Description
A1		
A2		Front knock sensor signal
A3		MAF sensor
A4		MAF sensor
A5		MAF sensor
A6		
A7		Cooling fan
A8		
A9		Fuel injector
A10		Fuel injector
A11		IAC valve
A12		+12 volt (switched ignition)
A13		Ground
A14		Front lambda sonde (feed)
A15	Y	TPS (out)
A16	G/W	TPS (in)
A17		Knock sensor common ground
A18	BR/B	Ground (sensor)
A19		Rear lambda sonde (ground)
A20		
A21	BL/Y	Camshaft position sensor
A22		Cooling fan
A23		Fuel injector
A24		Fuel injector
A25		IAC valve
A26		+12 volt (battery)
A27		+12 volt (?)
A28		Ground
A29		Rear lambda sonde (feed)
A30		Rear knock sensor signal
A31	Y/GY	Engine temperature sensor
A32		Front lambda sonde (signal)
A33		Front lambda sonde (ground)
A34		Rear lambda sonde (signal)
A35	G/Y	EGR temparature sensor
A36	Y/R	Camshaft position sensor
A37		
A38		Fuel injector
A39		EVAP valve
A40		EGR converter
A41		Fuel injector relay
A42		Ground
A43		

Pin number	Color	Description
B1		Vehicle speed sensor (feed)
B2		Automatic transmission control module
B3		Automatic transmission control module
B4		Automatic transmission control module
B5		To diagnostics socket (also to Aut-transmission control module)
B6	GY	A/C
B7		Check engine light
B8		Enable internal ECU ROM flashing when +12V is applied (results in +5V on EA pin on CPU)
B9		A/C Pressure sensor
B10		
B11		Ignition system power stage
B12		Automatic transmission control module
B13		
B14		
B15		
B16		
B17		
B18		
B19		
B20		Automatic transmission control module
B21		To tacho meter
B22		
B23		To engine temperature meter
B24		Automatic transmission control module
B25	BL/GY	A/C
B26		Automatic transmission control module
B27		To fuel pump relay
B28		A/C Pressure sensor
B29		A/C Pressure sensor
B30		
B31		
B32		Vehicle speed sensor (signal)
B33		
B34		
B35		
B36		To diagnostics socket
B37		
B38		
B39		To trip computer
B40		To A/C relay
B41		Turbo control valve (turbo models only)
B42		Automatic transmission control module
B43		

Sequential fuel injection (SFI) system, Motronic 4.3 (B 5204 T, B 5234 T/T5)

GN GN
GR GN
GR-OR GR-R GR-SB
GN GN GN
GR-W

8/6–8/10
24/15
4/28
31/4B
6/31
GR
SB
SB
C/BA
P
SB

20/3-7
17/7
4/46
BN GR BL-R
BL-BN
14
10
9
W
6
Y-BN
4/7

7/53
7/79
GR-OR
BN
OR-W

31/51
31/50
11/1
11/2
2/30
11/31
11/15 11/33
4/3
2/31
2/23
15/2

8/17
GN-BN GN
C/EA
6/29
7/47
SB
GN
C/EC
8/3
7/50
31/55
2/11

2/32
R
OR-SB
GN

8/18
BL-GN
GR-SB
BL-BN
GN
8/28
31/4
31/52
31/44

7/15
7/82
C/EP
GN-GR GN-SB
SB GN
SB
W
GN-SB
BN
10/106
Y-SB
GR-R
W-SB
BL-R
GN-GR
BN-W
5/1
A
B
C/EQ
BN
W

7/24
1 VO
2 P-SB
7/23
1 P
2 P-SB
31/32
31/33
SB
BN
7/16
Y-GR
BN-SB
C/EG
4/44
BL-R
Y-W
R 15
BL
GN-R
BN-W
GN
W
7/17

24/2
BL-GR
GN-GR
Y-BN
Y-VO
BL-SB
BN

7/54
7/37
C/EN
BN-SB
GN-Y
2
1
3
BN-SB
Y
OR-W
24/13
BL-R
W-SB
GN-GR
GR-R
Y-SB
17/13
BN-W
3/56
3/54
4/30
2/22
31/15
24/12 24/11

8/5
C/EJ
7/25
BL-SB
BL-W
GN
BL (MAN) R-SB
R-SB (MAN) BL (AUTO)
Cyl 2 Cyl 1 Cyl 3 Cyl 4 4/44 Cyl 5
20/2
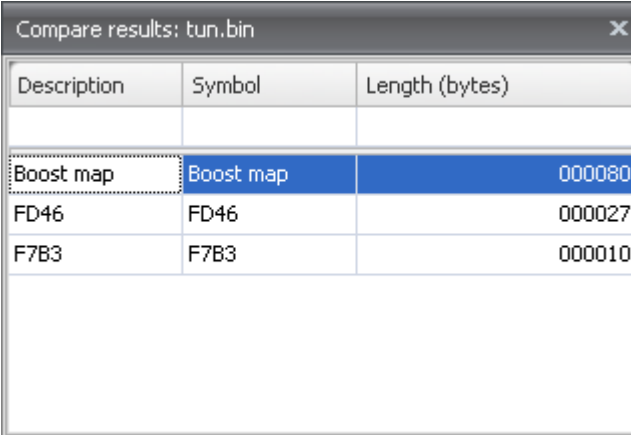C/EH
7/21
Y-R
BN-SB
BL-Y
24/4
2/22

63

Software

Once we download the data from the ECU – either with a flash programmer or by hooking up a K-line interface we can load the file into Motronic Suite (ref: http://trionic.mobixs.eu/Motronic/Motronic.msi)

Details on how the software works can be found in appendix I.

We can see the most important maps being automatically detected and we can change them to our likings. Be careful though, you need to know what you are doing. The software can generate a differences list between two files as well and if we compare a stock file to a tuned file we can see only a few maps get edited normally.

Compare results: tun.bin		
Description	Symbol	Length (bytes)
Boost map	Boost map	000080
FD46	FD46	000027
F7B3	F7B3	000010

Figure 1: differences between stock and tuned maps
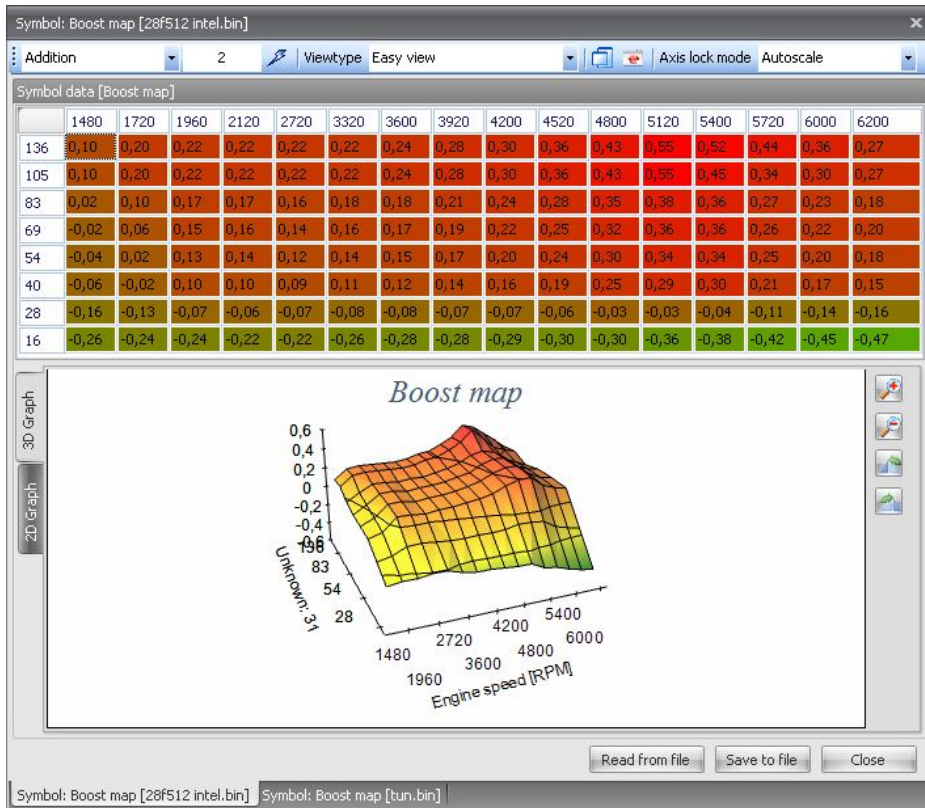
Figure 2: stock boost map

Figure 3: Tuned boost map

Software conversion factors

The software contains a lot of information for which we have to know the conversion factors to use to be able to translate them into units that we use on a daily basis. This is a shortlist of the most used factors.

36 Battery voltage *0.0704 Volt
38 Engine Coolant Temperature (ECT) *1-80 < -50 degrees Celsius
3B Engine speed *40 RPM
40 Internal load signal *0.05 msec
4C Mass Air Flow (MAF) sensor signal *0.01952 Volt
55 Ignition advance *(-0.75)+78 Degrees
BC Turbo duty cycle *0.391

Complete document here: http://trionic.mobixs.eu/Motronic/M4.3/M43-850-scaling.pdf

Software information data

In the software, the identifiers are stored as well about HW revision, SW version, volvo partnumbers etc. This data is stored in ASCII in the binary file and looks something like this:

```
0000ebc0h: 04 03 03 02 03 03 03 01 04 01 03 04 02 03 04 03 ; ................
0000ebd0h: 30 32 36 31 32 30 34 31 33 34 31 30 33 37 33 35 ; 0261204134103735
0000ebe0h: 38 35 38 36 39 31 32 35 33 32 39 FF FF FF 49 4D ; 85869125329ÿÿÿIM
0000ebf0h: 31 39 31 32 35 33 32 39 20 20 30 30 31 1E 1E 1E ; 19125329  001...
0000ec00h: 1E 1A 18 14 12 10 0F 0E 0E 0E 0E 0E 0E 0E 00 25 ; ...............%
```

In which
0261204134 is the hardware ID
1037358586 is the software ID
9125329 001 is the Volvo partnumber

And there's more readable information in the file as well:

```
0000fef0h: 26 20 1A 13 13 13 13 13 26 20 1A 13 13 13 13 13 ; & ......& ......
0000ff00h: 56 38 34 35 2F 31 2F 4D 34 2E 33 2F 31 39 2F 31 ; V845/1/M4.3/19/1
0000ff10h: 31 37 2E 35 30 2F 44 41 4D 4F 53 32 35 2F 2F 32 ; 17.50/DAMOS25//2
0000ff20h: 35 30 30 41 45 30 31 2F 30 34 30 36 39 36 2F FF ; 500AE01/040696/ÿ
0000ff30h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ; ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
```

45/1/M4.3/19/117.50/DAMOS25//2500AE01/040696
In which the latter string is the date of software build.
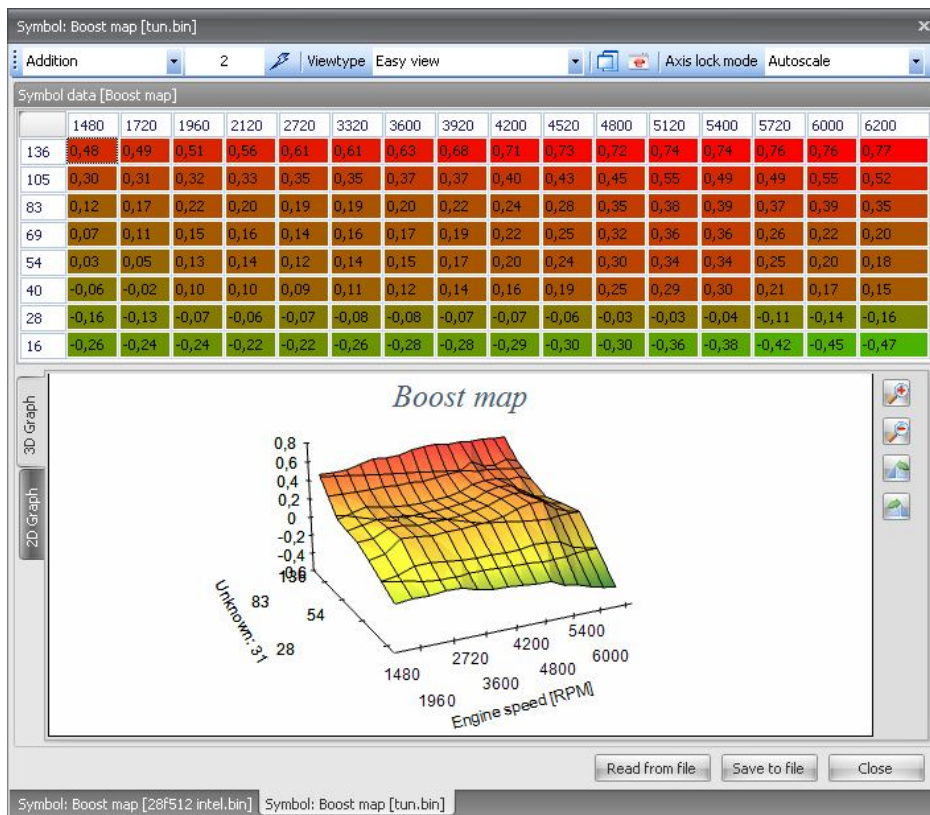
Reading the code

To be able to understand the software better we'll need to dive into the world of assembler language. This is a sort of intermediate between understandable human language and the operation codes used by the microprocessor. Once we can read the assembler language (assembly for short) we can track all the things the microprocessor is told to to when the program is running. This is very valuable information because we don't have first hand information from either Bosch or Volvo that can tell us in details what the ECU does.

We convert the binary file into assembly language we need to disassemble the file. We can do that by running the disassembler in Motronic Suite, by running the disassembler manually or by using a seperate program like IDAPro to do it for us. A seperate disassembler can be found here in the website.

Disassembler D52 http://trionic.mobixs.eu/Motronic/M4.3/d52.exe

Once we disassemble the binary file we have an file containing the complete assembly listing in which we can start to explore and understand the internal workings of M4.3.

Binary example file Volvo 850R: http://trionic.mobixs.eu/Motronic/M4.3/850R.bin

Assembler listing for this file: http://trionic.mobixs.eu/Motronic/M4.3/850R.asm

Maps and variables

Determining the location and type of maps and variables in the M4.3 binaries is quite a hassle. To be able to detect the available maps we have to do some tricks and make a couple of assumptions in the algorithm used. This chapter will describe – in detail – what the Motronic Suite software does to fetch the maplist from the file.

Getting the lookup table index

First we need to lookup the index in the file at which the lookup table is located. This lookup table consists of addresses in the file which we can use to determine axis and map information. Do find the correct index we look for a certain byte sequence in the file. M4.3 has a leading sequence of 4 bytes that always seem to be the same, 0x00 0x02 0x05 0x07. The picture below shows the data found in a certain M4.3 file.

```
0000df60h: 0D 01 02 04 01 02 09 0A 01 01 00 02 04 00 02 00 ; ................
0000df70h: 00 00 02 05 07 E9 F1 E9 F7 EA 11 EA 2B EA 35 EA ; .....éñé÷ê.ê+ê5ê
0000df80h: 3D E9 F1 EA 43 EA 4F EA FB EB 56 EB 64 EB 6E EB ; =éñêCêOêûëVëdënë
0000df90h: 7C EB 86 EB 90 EB 98 EB A2 EB B0 EB BE EB CC EB ; |ë†ëŒë¯ëœë°ë¾ëÌë
0000dfa0h: DF EB F2 EC 04 EC 16 EC 24 EC 2A EC 34 EC 3E ED ; ßëòì.ì.ì$ì*ì4ì>í
0000dfb0h: 62 ED 84 ED 8E ED 98 ED A2 E9 F1 ED BE ED C8 ED ; bí„íží˜í¢éñí¾íÈí
0000dfc0h: CE ED D8 ED E0 EE 4B EE 51 EE 87 EF AB F0 CF F1 ; ÎíØíàîKîQî‡ï«ðÏñ
0000dfd0h: F3 F1 FD E9 F1 E9 F1 F2 07 F2 11 F2 45 F2 6B F2 ; óñýéñéñò.ò.òEòkò
0000dfe0h: C0 F2 D0 F2 87 F2 DA F2 E4 F2 EE F2 F8 F3 04 F3 ; ÀòÐò‡òÚòäòîòøó.ó
0000dff0h: 0E F3 18 FA C9 FB 04 FB 3F FB 7A FB 84 FB 8E FB ; .ó.úÉû.û?ûzû„ûŽû
0000e000h: 9A FB DC F3 22 F3 30 F3 64 F3 B8 F4 3D F4 45 F4 ; šûÜó"ó0ódó¸ô=ôEô
0000e010h: 4D F4 A1 F4 A7 F4 AF F4 D1 F4 F3 F4 FB F5 1D F5 ; Mô¡ô§ô¯ôÑôóôûõ.õ
0000e020h: 3F F5 61 F5 83 F5 A5 F6 0E F6 77 F6 E0 F7 49 F7 ; ?õaõƒõ¥ö.öwöà÷I÷
0000e030h: F1 F8 03 F8 57 F8 69 F8 73 F8 7D F8 87 F8 91 F7 ; ñø.øWøiøsø}ø‡ø'÷
0000e040h: B2 F7 D9 F7 E5 F8 AD F8 B7 F9 49 F9 53 F9 59 F9 ; ²÷Ù÷åøø·ùIùSùYù
0000e050h: AD F9 CF F9 D7 F9 DF F9 E7 F9 EF F9 F7 F9 FF FA ; ùÏù×ùßùçùïù÷ùÿú
0000e060h: 07 FA 0F FA 17 FA 1F FA 27 FA 47 FA 4F FA 57 FA ; .ú.ú.ú.ú'úGúOúWú
0000e070h: 5F FA 67 FA 6F FA 77 F3 C2 F4 16 FA 7F FA 89 FA ; _úgúoúwóÂô.úú‰ú
0000e080h: 93 FA 9D FA A7 FA B1 FA BF 32 02 06 07 00 28 6F ; "úú§ú±ú¿2....(o
0000e090h: 2C 05 01 01 03 06 01 08 01 0A 01 32 01 01 06 06 ; ,..........2....
0000e0a0h: 06 28 05 3C 40 04 4C 01 60 8C 58 FE FE 03 02 02 ; .(.<@.L.`ŒXþþ...
0000e0b0h: B6 41 64 14 C3 DB B0 39 73 05 C8 7B 25 15 92 BE ; ¶Ad.ÃÛ°9s.È{%.'¾
0000e0c0h: 98 1E E4 A3 64 80 62 45 10 0F 00 20 09 A0 BB E3 ; ".ä£d€bE... .»ã
0000e0d0h: 39 19 08 0C 48 78 CD 5A 2B 32 3F 21 22 02 14 08 ; 9...HxÍZ+2?!"...
```

Reading the lookup table

The data directly after 0x00 0x02 0x05 0x07 are addresses of axis and maps, 2 bytes at a time. So, the first address we find is 0xE9F1 and the second one is 0xE9F7. We keep reading addresses and storing them in a list until we reach a 2 byte value that is smaller than 0xE000 (so, in fact we assume that there are no maps and axis located in the memory section < 0xE000).

Validating the entries in the table

Now we have a list of addresses with which we can work. For each address that we found we validate the data found at that address. For example we will look at the first address we found earlier, 0xE9F1.

```
0000e9f0h: 00 38 02 41 61 80 80 04 0C 07 0D 14 1B 1F 20 1C ; .8.Aa€€....... .
0000ea00h: 17 12 0D 0A 18 1E 28 32 3C 46 50 5A 64 6E 78 82 ; ......(2<FPZdnx,
```

The first two bytes tell us a lot. If the symbol is an axis (support point list) the first byte is the identifier which tells us the type of data it consist of. In this case, the first byte is 0x38 which translates to coolant temperature. The second byte is the length of the axis, in this case only two bytes. The Motronic Suite software assumes (thin ice!) that symbols found that start with a value >= 0x03 and <= 0x99 AND that have a length < 32 are axis symbols. After validating the symbol the software tries to determine any trailing axis symbol right after the symol that was just validated. In our example it looks at address 0xE9F5 an again validates this offset to be an axis (using the same method). We find identifier 0x80 and length 0x80 which would not classify this data as being an axis because the length exceeds 32 bytes.

This procedure is done for all addresses found in the lookup table and the software stores the found axis in a list containing the addresses and the data for the axis.

Determining the map data

What comes next is a little trick: we sort all the axis by address and validate the empty spaces in between the consequtive axis. If we see a "gap" in between the axis, we assume that there is map data in between that can be connected to the leading axis information (the axis that comes before the mapdata in the file). If we find two axis (and X and and Y axis) just before of the mapdata, the map is assumed to be 3D. If we only find one axis just before the mapdata, we assume the map is 2D.

Labeling the maps

This is possibly the part that has the most assumptions in it. Some maps are determined by their content and this will not always work, especially when the file has been tuned.

Ignition and VE maps

The ignition and VE map have a fixed length of 256 bytes in which the size is 16x16. Since there are three different ignition maps and only one VE map it is pretty tough to determine which map is which based only on their size. So, the content of the 256 bytes maps are evaluated and the map that has a lot of values around 127 (lambda 1) is said to be the VE map. The other 256 byte maps are the ignition maps in which the sequence in which they appear in the file (based on the address) determine which ignition map it is. The first one is the normal ignition map, the second one is the warmup ignition map and the third and last one is the knock ignition map.

Boost map

The boost map has a length of 128 bytes and since there is only one map in the file with this length it is pretty easy to label this one and the boost map.

MAF to load map

The MAF to load conversion map has a length of 64 bytes and the x axis is of type 0x30. Since there is only one map with these properties it is easy to label it as such.

Overboost map

The overboost map has a length of 64 bytes and the x axis is of type 0x3B. Since there is only one map with these properties it is easy to label it as the overboost map.

Finding the engine speed limiters

This is actually pretty hard to do, because there's no indicators as far as I know to determine the exact location in the file. Obviously something had to be worked out so here is the method used in Motronic Suite at the moment.

The reversed code shows this piece of code that uses a pointer to the rpm limiter:

```
code:00000FB5          mov          DPTR, #0xE0E8
code:00000FB8          mov          A, #4
code:00000FBA          cjne         A, RAM_3E, code_FC2
code:00000FBD          mov          A, #0x86
code:00000FBF          cjne         A, RAM_3D, code_FC2
```

In hex this looks like:

```
00000fb0h: F7 18 19 E6 F7 90 E0 E8 74 04 B5 3E 05 74 86 B5 ; ÷..æ÷□àèt.µ>.t†µ
00000fc0h: 3D 00 B3 92 26 40 1C 74 05 93 B5 3E 06 74 04 93 ; =.³′&@.t.“µ>.t.“
```

So we search the file for bytes sequence 0x74 0x04 0xB5 0x3E 0x05 0x74 0x86 0xB5 0x3D 0x00. The two byte in front of this sequence is the pointer to the RPM limiter.

There also seems to be a second RPM limiter in the file that is located 3 bytes further than the first one. This one is only 1 byte long and should be multiplied by 40 to get a valid RPM value. This second limiter is also handled and updated by Motronic Suite.

```
code:000088FE          mov          DPTR, #0xE0E8
code:00008901          mov          P2, #0x60
code:00008904          mov          R0, #0x55
code:00008906          movx         A, @R0
code:00008907          mov          RAM_2E, A
code:00008909          mov          R0, #0x57
code:0000890B          mov          A, #3
code:0000890D          movc         A, @A+DPTR
code:0000890E          cjne         A, ENGINE_SPEED, code_8911
```

Finding the vehicle speed limiter

This is actually pretty hard to do, because there's no indicators as far as I know to determine the

The reversed code shows this piece of code that uses a pointer to the speed limiter:

```
code:0000496A          mov          DPTR, #0xE103
code:0000496D          clr          A
code:0000496E          movc         A, @A+DPTR
code:0000496F          mov          B, @R1
code:00004971          cjne         A, B, code_4974
code:00004974 code_4974:
code:00004974          mov          RAM_2A.0, C
code:00004976          ret
```

In hex this look like:

```
00004960h: FE 90 61 B6 E0 44 02 F0 79 B8 90 E1 03 E4 93 87 ; þ□a¶àD.ðy□á.ä`‡
00004970h: F0 B5 F0 00 92 50 22 75 A0 60 78 77 20 24 04 E4 ; ðµð.'P"u `xw $.ä
```

So we search the file for byte sequence 0xE4 0x93 0x87 0xF0 0xB5 0xF0 0x00 0x92 0x50 0x22 0x75 0xA0. The two bytes in front of this sequence is the pointer to the vehicle speed limiter.

Communication with the ECU

Connection diagram
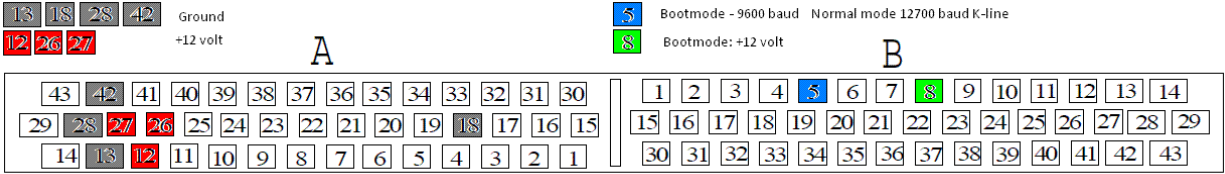
There are two methods of communication that can be used with a M4.3 ECU.
- Normal mode communication
- Boot mode communication

Normal mode communication

Normal mode is used for reading data from the ECU while it is in its operational state. Reading live data and the contents of the flash file are procedures that are carried out in normal mode.
To activate normal mode communication we need to connect a K-line interface to the ECU on pin B5 and after the wake-up procedure communication can commence at 12700 baud.

Wakeup procedure for normal mode

To be able to communicate in normal mode, the ECU needs to be aware of the fact that there is a diagnostics device connected to pin B5. To let the ECU know we need to send a 0x10 byte to the port at 5 baud (!). After a correct wakeup byte on the B5 pin we will receive a response from the ECU at 12700 baud. This response will be 0x55 0xAB 0x02 in which the 0x55 is the acknowledge and the 0xAB and 0x02 are the keywords used to communicate with a M4.3 ECU. After reception of this sequence we need to send an acknowledge message to the ECU which is the inverted last keyword which will be 0xFD.

Normal mode: KWP71

After the wakeup procedure, communication with the ECU takes place in the KWP71 protocol. This protocol is standarized and therefore it will not be discussed within this document.

Bootmode communication

Boot mode communication is only useful for flashing the ECU with a new firmware version. This is a special mode which is indicated to the ECU by pulling pin B8 to +12V before the ECU starts (boots). So, you will need to have the ECU powered down and apply +12V to B8 before the ECU is powered on. The ECU will now run a special boot loader program which resides in internal ROM and allows us to reprogram the ECU. After starting in bootmode communication can commence on pin B5 at 9600 baud with a K-line interface.

A correctly formatted command line should result in an acknowledge (ACK)
0x02 0x30 0x59 0x30 0x37 0x03 0x62
Wrong commands result in a negative acknowledge (NACK)
0x02 0x30 0x59 0x30 0x38 0x03 0x62

Programming sequence

The programming sequence for M4.3 looks like any other ordinary reflash sequence.

- Establish and verify communication with the ECU
- Erase the flash chip
- Program the flash chip
- Verify sequence and disconnect

Verifying communication

To verify a valid connection between application and ECU a dummy command is issued on the serial port. Send some valid sequence like 0x3A 0x00 0x00 0x00 0x00 0x00 0x00 0x00
This should result in an ACK returned from the ECU.
The 8-bit sum of the numbers behind 0x3A should always be 0x00, this functions as a checksum.

Erase the flash chip

To erase the flash we need to send command 0x01 with a valid keycode for access to the ECU. The access code is 0x11 0x22 0x33 0x44 0x55. The final command for erasing the flash is:
0x3A 0x01 0x11 0x22 0x33 0x44 0x55
Since erasing the flash takes some time and the application controlling the ECU should not freeze and should be able to deliver some feedback to the user the ECU delivers two responses to this message, one is issued when it starts to erase the flash and one is issued after the erase procedure has finished.
Start erase: 0x02 0x30 0x59 0x30 0x39 0x03 0x62
Finished erase: 0x02 0x30 0x59 0x30 0x35 0x03 0x62

Program the flash chip

After erasing the flash we have to send the byte sequences to program the flash with new values. This is actually the new binary file that we are sending to the bootloader so it can program this into the erased flash. Since we cannot send the entire binary file at once (because of limits to the receive buffer in the ECU) we have to send small blocks of data at a time and repeat this procedure until all data is processed.

Each command contains up 32 bytes of data to be programmed and the address it should be programmed at in the flash.

0x3A 0x20 0xaddr1 0xaddr2 0xaddr3 [32 bytes of program] [checksum, 1 byte]

A typical sequence would be (0x indicators are omitted here for easier reading, all data is in hex though)

3A 20 00 00 00 02 7D 44 02 2A DE FF FF FF FF FF 02 2A E6 FF FF FF FF FF 05 17 32 FF FF FF FF FF 02 1D 97 FF FF FF 0E

command	3A 20
address	00 00 00
data	02 7D 44 02 2A DE FF FF FF FF FF 02 2A E6 FF FF FF FF FF 05 17 32 FF FF FF FF FF 02 1D 97 FF FF
checksum	0E

Please note that the address bytes are structured a bit odd. The file is 0xFFFF bytes long so normally we would assume that the address in the command would range from 00 00 00 upto 00 FF FF but this is not the case. We need to have the high order by in addr1 and the low order by in addr2, while we keep addr3 filled with 0x00. Programming at position 0xCAE0 would yield a command that starts with

3A 20 CA E0 00

Verify sequence and disconnect

After we have send all the data containing the new binary to the ECU we need to make sure that the ECU understood what we send and if it succeeded in programming the flash chip. This is important because the timing of the programming sequence is key and there might be disturbances when windows tries to do silly things.

To verify the operation we send these two commands and we wait for a response from the ECU.

3A0000000000

3A30303030303030314646

If the response is
02 30 59 30 34 03 xx we have a positive indication from the ECU.
If the response is some other command like
02 31 59 30 31 03 xx or
02 31 59 30 38 03 xx we have a negative indication from the ECU and we should retry programming.

Boot rom information

The bootrom can be download here: http://trionic.mobixs.eu/Motronic/M4.3/M43_rom.bin
And the corresponding assembly listing: http://trionic.mobixs.eu/Motronic/M4.3/bootrom.asm

This is given for reference only so you can explore the bootrom on you own.

Appendix I: Reverse engineering the binary

The Motronic 4.3 binary files are not extremely hard to reverse engineer. The 8051 based processor in it means that there are several freeware disassemblers available to work with. The thoughest part is to extract all the map and axis information from the file and to automatically determine which map means what. Currently all maps are extracted and some are named by the software in the extraction process like the ignition, main fuelling and boost maps.

The extraction process works as follows.

1. We look through the file for sequence 0x00 0x02 0x05 0x07
2. After that we read 2 bytes at the time as integer 0xED 0x4C = 0xED4C = 60748 which is an axis address in the file. We do this until the first read byte is smaller than 0xE0.
3. We now have a first list of addresses to process. For each found address we read two bytes at that particular address in the file. For the given example fo address 0xED4C we would read f.e.: 0x04 0x0C. The first byte is the identifier (which type of data it concerns) and the second byte is the length of the data for this axis. So, we have identifier 0x04 and length 0x0C. Now we continue to read 0x0C (=12) bytes from the current location in the file. We read f.e. 0x07 0x0D 0x14 0x1B 0x1F 0x20 0x1C 0x17 0x12 0x0D 0x0A 0x18.
4. Motronic 4.3 has coded axis values, so we need to decode this information to get to the "real" values that correspond to units we use every day. This decoding is done as follows. We take the last value in the list (0x18) and calculate the maximum axis value with this formula: (256 – last_value) * correction factor. The correction factor for axis with ID 0x04 is 1 (ref: correction factor document), so the maximum value would turn out to be 256 – 0x18 = 256 – 24 = 232. Next we take the second to last value from the list (0x0A). The real value is again calculated as follow previous_real_value – (value * correction factor). This results in 232 – (0x0A * 1) = 232 – 10 = 222. The next values are calculated in the same way. The next value is 0x0D which means 222 – (0x0D * 1) = 222 – 12 = 210. We calculate all values in the list like this and we end up with:
   a. 256 – 0x18 = 256 – 24 = 232
   b. 232 – (0x0A * 1) = 232 – 10 = 222
   c. 222 – (0x0D * 1) = 222 – 12 = 210
   d. 210 – (0x12 * 1) = 210 – 18 = 192
   e. 192 – (0x17 * 1) = 192 – 23 = 169
   f. 169 – (0x1C * 1) = 169 – 28 = 141
   g. 141 – (0x20 * 1) = 141 – 32 = 109
   h. 109 – (0x1F * 1) = 109 – 31 = 78
   i. 78 – (0x1B * 1) = 78 – 27 = 51
   j. 51 – (0x14 * 1) = 51 – 20 = 31
   k. 31 – (0x0D * 1) = 31 – 12 = 19
   l. 19 – (0x07 * 1) = 19 – 7 = 12

   Our calculated axis now results in: 12 19 31 51 78 109 141 169 192 210 222 232

5. We now also – for the sake of understanding – also calculate an axis the has the load value (which is calculated in ms in Motronic) as identifier (0x40). M4.3 has a correction factor of 0.05 for this axis. We read 0x40 for ID and 0x10 for length. The data is 0x07 0x07 0x08 0x08 0x0A 0x0A 0x0A 0x0A 0x0D 0x0D 0x0D 0x0D 0x0D 0x0D 0x0D 0x55. Now we calculate in the same way as in point 4.

    a. 256 – 0x55 = 256 – 85 = 171 * 0.05 = 8.55
    b. 8.55 – (0x0D * 0.05) = 8.55 – 0.65 = 7.90
    c. 7.90 – (0x0D * 0.05) = 7.90 – 0.65 = 7.25
    d. 7.25 – (0x0D * 0.05) = 7.25 – 0.65 = 6.60
    e. 6.60 – (0x0D * 0.05) = 6.60 – 0.65 = 5.95
    f. 5.95 – (0x0D * 0.05) = 5.95 – 0.65 = 5.30
    g. 5.30 – (0x0D * 0.05) = 5.30 – 0.65 = 4.65
    h. 4.65 – (0x0D * 0.05) = 4.65 – 0.65 = 4.00
    i. 4.00 – (0x0A * 0.05) = 4.00 – 0.50 = 3.50
    j. 3.50 – (0x0A * 0.05) = 3.50 – 0.50 = 3.00
    k. 3.00 – (0x0A * 0.05) = 3.00 – 0.50 = 2.50
    l. 2.50 – (0x0A * 0.05) = 2.50 – 0.50 = 2.00
    m. 2.00 – (0x08 * 0.05) = 2.00 – 0.40 = 1.60
    n. 1.60 – (0x08 * 0.05) = 1.60 – 0.40 = 1.20
    o. 1.20 – (0x07 * 0.05) = 1.60 – 0.35 = 0.85
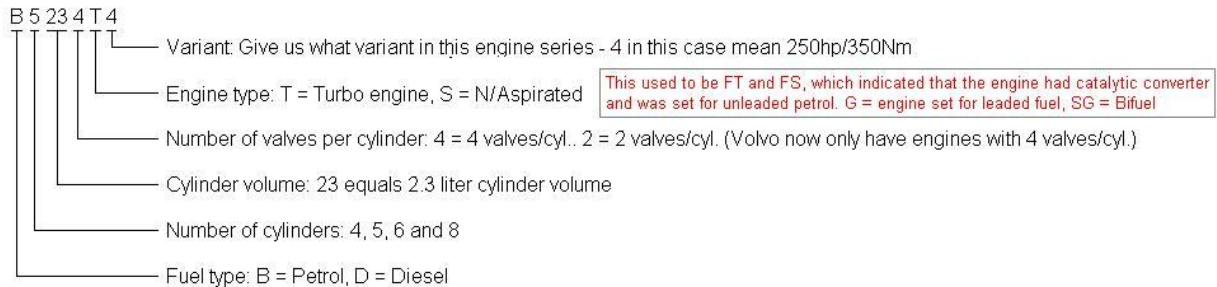   p. 0.85 – (0x07 * 0.05) = 0.85 – 0.35 = 0.50

So, our final axis is this:

0.50 0.85 1.20 1.60 2.00 2.50 3.00 3.50 4.00 4.65 5.30 5.95 6.60 7.25 7.90 8.55

Appendix II: Partnumber and engine information

The ECU partnumbers and engine code information is important when starting your tune. This appendix will give you handles on where to start.

Engine code

B 5 23 4 T 4

Variant: Give us what variant in this engine series - 4 in this case mean 250hp/350Nm

Engine type: T = Turbo engine, S = N/Aspirated

This used to be FT and FS, which indicated that the engine had catalytic converter and was set for unleaded petrol. G = engine set for leaded fuel, SG = Bifuel

Number of valves per cylinder: 4 = 4 valves/cyl.. 2 = 2 valves/cyl. (Volvo now only have engines with 4 valves/cyl.)

Cylinder volume: 23 equals 2.3 liter cylinder volume

Number of cylinders: 4, 5, 6 and 8

Fuel type: B = Petrol, D = Diesel

Example:

B5234T — Petrol 5 cylinder 2.3liter 4-valves/cylinder with turbo (225hp/300Nm)

B5234T4 — Petrol 5 cylinder 2.3liter 4-valves/cylinder with turbo (variant 4, in this case 250hp/350Nm)

B6294T — Petrol 6 cylinder 2.9liter 4-valves/cylinder with turbo (272hp/380Nm)

B6294S2 — Petrol 6 cylinder 2.9liter 4-valves/cylinder N/Aspirated (variant 2, in this case 196hp/280Nm)

B5254T4 — Petrol 5 cylinder 2.5liter 4-valves/cylinder with turbo (variant 4, in this case 300hp/400Nm)

B5244T3 — Petrol 5 cylinder 2.4liter 4-valves/cylinder with turbo (variant 3, in this case 200hp/285Nm)

B5244S2 — Petrol 5 cylinder 2.4liter 4-valves/cylinder N/Aspirated (variant 2, in this case 140hp/220Nm)

B4204T3 — Petrol 4 cylinder 2.0liter 4-valves/cylinder with turbo (163hp/240Nm)

D5252T — Diesel 5 cylinder 2.5liter 2-valves/cylinder with turbo (140hp/290Nm)

D5244T — Diesel 5 cylinder 2.4liter 4-valves/cylinder with turbo (163hp/340Nm)

ECU partnumber	Car model	MY
0 261 203 074	850 2.3 T5 Automatic	1994-1995
0 261 200 549	850 2.3 T5 Manual	1994-1995
0 261 203 627	850 T5R Automatic (Euro spec)	1995-1996
0 261 203 628	850 T5R Automatic (US spec)	
0 261 203 626	850 T5R Manual	1995-1996
0 261 204 134	850R Automatic	1996-1997
0 261 204 225	850R Manual	1996-1997
0 261 203 852	850 2.3 T5 Automatic	1996-1997
0 261 203 851	850 2.3 T5 Manual	1996-1997
0 261 203 962	850 2.0 T5 Automatic	
0 261 203 189	850 2.0 GLT Automatic	

Appendix II: Building a high speed K-line interface

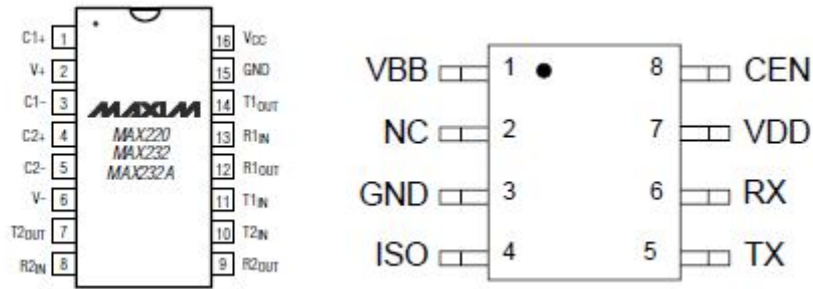The following information is sourced from skpang.co.uk.

Schematic
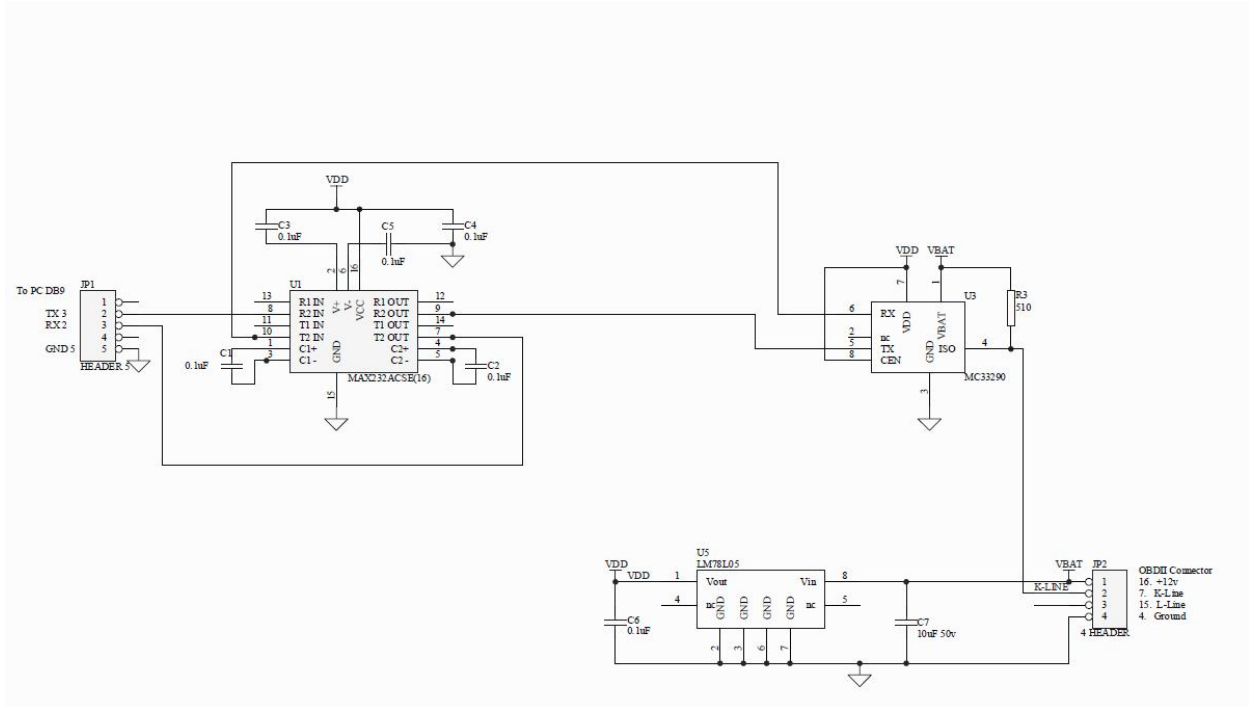
Parts information

Figure 4: MAX232 and MC33290